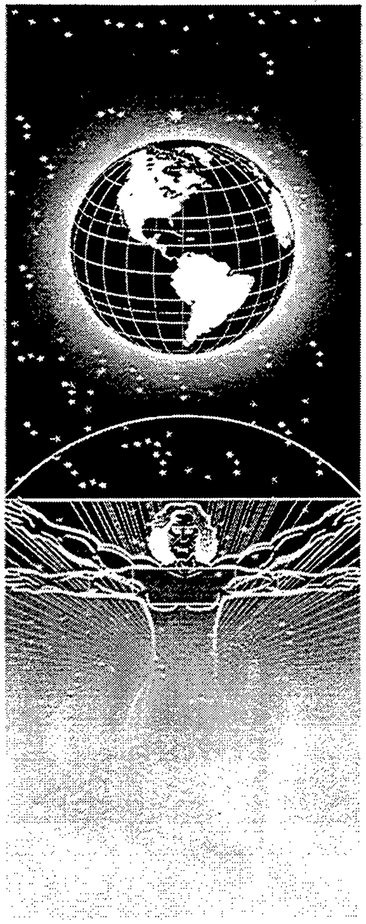
AFRL-HE-WP-TR-1998-0062



UNITED STATES AIR FORCE RESEARCH LABORATORY

Capturing Logistics Data from Simulations: DEPTH Technical Order Generation

K. W. Lane & B. G. Metcalf

Hughes Missile Systems Company
P.O. Box 11337
Tucson, Arizona 85734

April 1998

Interim Report for the Period February 1992 to October 1997

Human Effectiveness Directorate
Crew Survivability and Logistics Division
Sustainment Logistics Branch
2610 Seventh Street
Wright-Patterson AFB OH 45433-7901

NOTICES

DISCLAIMER

This Technical Report is published as received and has not been edited by the Air Force Research Laboratory, Human Effectiveness Directorate.

TECHNICAL REVIEW AND APPROVAL

AFRL-HE-WP-TR-1998-0062

This report has been reviewed by the Office of Public Affairs (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Lt Col, DEPUTY

For THOMAS J. MOORE, Chief
Crew Survivability and Logistics Division
Air Force Research Laboratory

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
April 1998

3. REPORT TYPE AND DATES COVERED
Interim - February 1995 - October 1997

4. TITLE AND SUBTITLE
Capturing Logistics Data from Simulations: DEPTH Technical Order Generation

5. FUNDING NUMBERS
C - F33615-91-C-0001
PE - 63106F
PR - 2940
TA - 03
WU - 05

6. AUTHOR(S)
K. W. Lane & B. G. Metcalf

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Hughes Missile Systems Company
P.O. Box 11337
Tucson, Arizona 85734

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory, Human Effectiveness Directorate
Crew Survivability and Logistics Division
Air Force Materiel Command
Sustainment Logistics Branch
Wright-Patterson AFB OH 45433-7604

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-HE-WP-TR-1998-0062

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
DEPTH uses a human figure model to visualize man-machine interaction and receive on-line human factors information simulations; such as simulating maintenance evaluations. The primary focus of the study is to investigate how DEPTH can support the automated generation of technical manuals in order to streamline the USAF technical order general process. DEPTH can produce text describing maintenance task steps suitable for Interactive Electronic Technical Manuals (IETMs). The language initially produced will consist of simple instructions, by using commercial IETM authoring tools.

14. SUBJECT TERMS
Logistics, Technical Order, Human Factors, Training, Simulation, Interactive Electronic Technical Manuals (IETMs), DEPTH, Computer Aided Design (CAD), generating technical orders, Maintenance, Logistics Support Analysis Control Number (LCN), virtual systems

15. NUMBER OF PAGES
148

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

THIS PAGE INTENTIONALLY LEFT BLANK

PREFACE

This report documents the results of a study of technical order generation conducted as part of a logistics research and development program titled Design Evaluation for Personnel, Training, and Human Factors (DEPTH) (contract number F33615-91-C-0001), managed by the Air Force Research Laboratory, Logistics Sustainment Branch (AFRL/HESS), at Wright Patterson AFB, OH. DEPTH uses a human figure model to visualize man-machine interaction and receive on-line human factors information simulations. The primary focus of the study is to investigate how DEPTH can support the automated generation of technical manuals in order to streamline the USAF technical order general process.

TABLE OF CONTENTS

1 Introduction

This report documents the results of a Hughes Missile Systems Company study conducted for the Design Evaluation for Personnel, Training, and Human Factors (DEPTH) program. The study investigated how DEPTH can support the automated generation of technical manuals. The recommendations from the feasibility study were to conform to MIL STD-87269.

Although the study focused on specific software systems, the results are generally applicable. For example, the modeling and simulation software used by DEPTH (namely Transom Jack) was one of several systems available to simulate human activity. However, much of the facilities needed to create technical manuals had already been built into DEPTH, so using other systems would require more work. The study focused on "generating the human's physical requirements input for technical orders" where:

1. "Technical orders" (TOs) was interpreted to mean computer-driven Interactive Electronic Technical Manuals (IETMs)

2. "Capture human physical requirements" was interpreted to mean:
   - Identify maintenance tasks and the associated steps with the appropriate action verb and target syntax
   - Capture graphical data in the format required by IETMs including:
     - illustrations
     - animations (i.e., movies)
   - Capture attributes available from the computer-aided design (CAD) system or the human models that are required to define a task
   - Identify cautions and warnings when appropriate

The study was performed by two organizations under contract to the Air Force:

1. The *University of Pennsylvania* studied the linguistics of TOs and determined how Parallel Transition Networks or PaT-Nets (used in Jack to control motion of a human model) could be used to generate text.

2. *Hughes Missile Systems Company* examined a logistics database (MIL-STD-1388-2B compliant) and determined which records could be updated.

For this study we used two software packages (excluding embedded software packages):

1. <u>DEPTH</u> is a graphical simulation system used to create accurate representations of human/machine interaction. These capabilities allow engineers to evaluate maintenance activity and perform human-centered design analysis. This software consists of a set of modules that interact with other software applications including the Transom Jack human modeling system.

2. <u>Enhanced Automated Graphical Logistics Environment (EAGLE)</u> is a logistics database management system that can produce a variety of logistic products. This Windows-based system contains a collection of logistics management tools developed by Hughes. EAGLE is a validated MIL-STD-1388-2B database that automatically produces Class IV IETMs that comply with MIL-M-87268, MIL-D-87269, and MIL-Q-87269. EAGLE supports client-server distributed database environments.

During Phase V, the DEPTH task simulation capability was significantly extended and improved. This capability was fundamental for task creation. The capability to update appropriate fields in a relational logistics database was also added. Since EAGLE was already MIL STD-1388-2B compliant, Hughes and the government agreed that

demonstrating the ability to update this database was adequate to demonstrate feasibility to update similar systems. Together, these functions made it possible to demonstrate the feasibility of technical manual generation.

A benefit to this approach was that EAGLE client software already allowed field updates through a Structured Query Language (SQL) interface. When DEPTH needs to create or update a field, it needs only to generate an appropriate SQL statement to send to EAGLE

2 Steps For Generating Technical Orders

2.1 Task Analysis Process

The general steps followed to develop and analyze a maintenance task in DEPTH are illustrated below.

Define Environment
Create objects and define their attributes.

Define Tasks
Identify specific tasks to be performed.

Define Steps
Determine steps required to perform a specific task.

Create Simulation
Sequence the simulation segments needed to perform each step.

Run Simulation
Simulate task and display results for user evaluation.

This multiple step process is not particularly fast or simple. However since DEPTH is primarily used for design evaluation, we expect many of these steps to be accomplished by maintenance engineers for their process.

2.2 Short and Long Term Approaches

The steps in the maintenance task analysis process can be implemented in DEPTH using various approaches. Early in this feasibility study, it became clear that fully automating the process was not possible under the current DEPTH contract. Most significantly, all of the technology needed for a full implementation was not yet available. Even if all of the technology was available, we could not afford the man-hours required.

Therefore, a two pronged approach was proposed to address these limitations. We have attempted to identify short term implementations that require a relatively small investment in time and cost, and do not conflict with longer term possibilities. These long term goals either require more significant investments or technological advances.

A number of design guidelines and assumptions were made during the preparation of this document. For clarity, they are documented in section 2.2.2.

2.2.1 *Design Guidelines.*

A primary consideration of this design was to make this process as simple and easy as possible. Input required from the user was minimized and guidance was provided if data was required from the user. Input required from the user included:

- Identify objects and their properties in the simulation environment. This may be as simple as pointing to a file that contains a saved environment, but most likely is a more involved task, including creating a number of objects and identifying their properties.
- Evaluate and, if necessary, correct the object information generated
- Make judgments and decisions required to control and direct the analysis process
- Identify the specific tasks to be performed on an object or assembly
- Evaluate how well the simulations perform each maintenance task step
- Decide when the data collected is appropriate for transfer to the logistics (and IETM) database

Objects and their attributes in an environment file can originate from CAD systems, the user, or DEPTH itself. CAD data is usually limited to geometry but the user can add attributes to this data including motion axes, temperature and weight. All of this information can be stored into a DEPTH environment file. DEPTH provides the following capabilities:

- Catalog and preserve the properties for all objects
- Provide meaningful defaults, selection lists, and contextual help
- Allow generated information to be changed
- When possible, capture information from the simulation rather than require the user to supply it

Most data intensive input should eventually be available from engineering systems such as CAD. Thus, the time consuming task of setting up an environment should eventually be automated with improved interfaces to CAD.

2.2.2 *Assumptions*

The following items are expected to be available in electronic form from CAD systems in the future, either through the system itself or through translator software:

- Geometric positioning of objects
- Fastener information (type, orientation, and objects attached by)
- Joints between objects (fixed and flexible)
- Assembly/subassembly hierarchies (object association with assemblies/subassemblies)

The following items will be available in electronic form from engineering design packages in the future:

- Object mass (or weight)
- Object hazard information such as temperature, noise, and materials

2.3 Environment Definition

Although importing CAD objects provides fast and accurate geometry, it currently does not provide attributes such as weight and relationships to other objects. DEPTH also allows the creation of basic geometric shapes such as boxes, spheres, and cones. Some predefined DEPTH objects, such as fasteners, humans, attachments, tool sites, grasp sites, and hazards, have attributes needed for the simulation. For example, attachment sites are predefined allowing automated placement of tools to fasteners.

2.3.1 Short Term

Not much can be done in the short term to relieve the user from defining attributes. Building full feature interfaces to specific CAD systems is expensive and not a long term solution. Most CAD software applications have new versions at least annually and external interfaces often must be rewritten. Certain attributes needed for maintenance simulation are not available in most CAD packages anyway. Silicon Graphics and other vendors provide free translators for CAD geometry.

2.3.2 Long Term

Standards in product definition exchange should eventually allow a fuller description of assemblies to be extracted from engineering systems including CAD. STEP – The Standard for the Exchange of Product Model Data – is one of the most prevalent standards. However, many the attributes needed for maintenance simulation are not currently supported. Those involved in simulating human activity need to ensure their needs are also addressed in these standards.

Eventually we expect most of the attributes needed (i.e., weight, operating temperature, connections, assembly) will not have to be defined by the DEPTH user. CAD systems and their interfaces will continue to evolve. Perhaps someday "products" will be cut and paste between applications just as we do with text. Another possibility is an integration of CAD and simulation software; eliminating the need to port data altogether.

2.4 Task Definition

2.4.1 Short Term

Once the environment is defined, a simulation script can be created to define the maintenance task . Many tasks relating to specific objects, tools, fasteners, or humans are provided by DEPTH. These predefined task simulations are referred to as motion models. Where possible, mouse clicks and list selections are preferred over text data entry. Motion models can also be created by the user. All information that defines the task is stored as part of the simulation file.

2.4.2 Long Term

All of the capabilities in the short term should be useful in the long term as well. Plus the standard task list should be expanded to address more maintenance scenarios. We believe that artificial intelligence (AI) will be used to automatically build the task hierarchy with minimal user specification. Various software technologies, including natural language understanding and agents, should make this possible. With a modest research and development effort, commercial AI packages could be integrated with DEPTH.

2.5 Step Definition

2.5.1 Short Term

The DEPTH user creates a task by sequencing a series of steps (motions). Logisticians and technical manual authors undergo a similar process for their analyses. DEPTH thus can provide a structured method to analyze and verify task steps. For example, to remove a wheel assembly the user would specify:

1. grab handle

2. push handle
3. pull handle
4. release handle
5. grasp tool
6. loosen lug-nuts

As the simulation runs, the animation plays and the result of each step is recorded. Problems with the simulated task can be corrected "on the fly" and saved as a simulation file for later use. The simulations can also be captured as a frame-based animation for playback on a personal computer.[1]

2.5.2 Long Term

Relationships defined for objects in the environment can be used to automatically provide a sequence for assembly or disassembly. Dr. Ranko Vujosevic demonstrated [RV95] that if the following relationships are identified, they can be used to define removal and installation steps:

- attached
- covered
- engaged
- connected

Also, Deneb Robotics, Inc. – who produces another system that simulates human activity – advertises the ability to define relative motions between objects in their products as a feature that allows more intelligent manipulation of "motion machines."

Relationship information for objects could be provided through DEPTH menus as part of the environment definition process. Information provided would be retained and the user would have the ability to change relationships as errors were discovered. There is some possibility that DEPTH could be extended to detect errors in the relationships and recommend simple corrections. But initially, the primary responsibility for creating this information would rest with the DEPTH user.

As CAD databases improve, object relationships could be identified as products are developed. The information would be imported as part of the DEPTH environment definition process either directly from the CAD database, or through a CAD translator program. The user would then have the ability to modify the relationships used in the environment through drop down menus and dialog boxes. The approach would require additional mechanisms to be developed to capture the required object relationship data as part of the preparation of engineering designs.

An advantage of this approach is that the maintenance task steps that involve assembly or disassembly can be "automatically" generated from the object relationships. Care would have to be taken to ensure that cognitive steps could be inserted where needed in the motion sequence.

2.6 Animation

Every task and step is associated with a simulation file. It should be noted that all tasks are simulation files but not every simulation file is a task. Simulation files can contain commands such as view changes and external calls that are not actually part of a real world task. A set of actions are attributed to DEPTH objects (including assemblies and subassemblies). For example, it doesn't make sense to *pull* a wrench but it makes sense to *pull* a box from an access panel. Conversely it is hard to imagine why anyone would torque a box but torquing a wrench is common.

[1] Silicon Graphics' MoviePlayer and Apple Computer's QuickTime formats are currently supported. MPEG format can also be created using a conversion program available from Silicon Graphics.

2.6.1 Short Term

Maintenance task step definition and animation will continue to be closely intertwined. The user will use drop down menus, dialog boxes, and mouse clicks to indicate the motion and targeted object for each step.

The appropriate motion model will be activated immediately and the result is provided interactively to the user. The user can see the animation action and judge the success of each incremental movement as it is performed. Errors are corrected by adding or deleting motions until a satisfactory result is obtained. The entire motion sequence can then be saved as a simulation file and run at a later time. Screens can be captured and movies can be made of the simulated activity.

2.6.2 Long Term

The relationship between task steps and motion models is established when the user identifies the specific tasks that will be associated with an object. It is expected that the specific task and its associated steps will be identified on an extension of the task definition menu and dialog box. Standard maintenance task steps would be associated with a default motion model name or a previously defined PaT-Net. When this is sufficient, the user will not be required to provide further information.

When the default motions are not appropriate, two alternatives are possible. First, the DEPTH user will be allowed to specify a different motion model to animate the task. When this is sufficient, the user will not be required to provide further information. Second, when there are no existing motion models that perform the motion, the user will be required to use task editing tools[2] and the current DEPTH simulation capability to address the problem. When these tools are used, fairly complex tasks can be created by combining them in an hierarchical fashion. If a motion model is not available to achieve the desired results, the PaT-Net editing tools can be used to build a motion model from scratch. However, the PaT-Net author must associate its name with the appropriate task step.

A relatively simple improvement would allow the user to walk the human figure models through a task with DEPTH and/or Jack commands if motion models were not available to perform the action. This motion could then be captured as a motion model.

2.7 Simulation

When a task is defined and all steps are identified, it must be evaluated by running it in the DEPTH simulation environment. The user controls this operation from the simulation environment menus. Each maintenance task motion is animated using motion models, saved DEPTH simulation files, and PaT-Nets. DEPTH captures the appropriate information in logs and reports that can be analyzed at the conclusion of the run. Any errors and omissions in the task can then be corrected by updating the saved motion sequence in the simulation file.

2.8 Database Interface

DEPTH interfaces to EAGLE through SQL statements directed to the database server. SQL queries allow DEPTH to select which fields need to be created or updated. This process requires some preparation however once the interface is set up, it can be used to automate the creation and validation of logistics data.

2.8.1 EAGLE Client Activity

The EAGLE client software runs on remote machines with a network connection to the server. It allows the database to be updated and control the production of technical manuals. The software can then produce reports, Test Requirements Documents, Time Compliance Technical Orders, Illustrated Parts Breakdown,

---

[2] Task editing tools can display a task as a graphical network or as a list of textual descriptions. A text-based editor was created for the DEPTH program. A graphical editor, however, was not developed in time to incorporate into the software before the end of the contract.

Any logistics database field or table that DEPTH can create or update can also be modified through the client software. Therefore, it was decided that DEPTH should primarily modify fields where information was readily available from the simulation. By automatically extracting information from the simulation, the user need not go through the laborious task of authoring the task steps. DEPTH could also be used to prompt the user for additional information about the task, but this would be a secondary design goal.

The logistics database must be initialized by EAGLE before DEPTH can interact with it. That is, the client must create the logistics database, populate it to the point that DEPTH can run, and verify updates to the logistics database.

The fields that DEPTH will update are documented in the graphical user interface storyboard for logistics elements. Most of the fields are part of the maintenance and test support equipment area. After the database is updated, EAGLE can be used as originally designed to add cognitive task definitions and steps as required and to continue the database population process with information that is not appropriate for DEPTH input. When the population process is complete, the client software executes and controls the IETM publication cycle.

2.8.2 EAGLE Server Activity

The server runs on a server machine with network connections to the clients. It allows a system administrator to define user names and passwords as required for DEPTH and each EAGLE client. More importantly, it provides software that accepts and validates SQL statements from EAGLE clients and the DEPTH software. Data is provided to or received from the client and Data Element Description (DED) items in the LSAR tables are modified, as appropriate, as the SQL commands are executed on the server.

2.8.3 DEPTH Activity

2.8.3.1 Workspace Preparation

When the results of a simulation are used to update the database, the DEPTH environment file must include information to interface with EAGLE. This is stored in the environment file for use in later sessions. This information includes:

- EAGLE server name
- DEPTH user identification on the server
- password, if required, on the EAGLE server
- name of the LSAR database being updated
- end item acronym code that uniquely identifies the system
- Task Control Number and Logistics Support Analysis Control Number (LCN) represent a functional or hardware breakdown of the system, the part name, and an alternate LCN

Next the DEPTH environment must be populated with the items required to evaluate and validate a specific maintenance task. DEPTH's *insert* command is used to add CAD objects, humans, fasteners, and tools to the environment. CAD objects are usually part of the system being evaluated or objects not available as standard DEPTH objects. Examples include subsystem components, special tools, or support equipment. Human models can be generated in various sizes, genders, and protective clothing. Tools and fasteners are required to help automate the motion sequences for maintenance tasks requiring assembly and disassembly.

To build a virtual system that behaves like a real system, the user should:

- create locations (sites) needed to interact with objects such as handle grasp locations
- precisely position objects not available from CAD such as fasteners and cables
- create attachments or constraints between items

- establish properties such as mass and temperature

Information used to identify the fields to be updated in the EAGLE database is collected as each item is inserted into the DEPTH environment. At this point, the user is ready to create a simulation by selecting the Evaluate Simulation menu entry.

2.8.3.2 Simulation

When an environment file is opened, dialog boxes are provided to collect and verify definition information about maintenance tasks, subtasks, and task elements. When a LSAR maintenance task, subtask, and task element has been defined, a motion model library is made available through a set of user dialog boxes. The standard motions in these libraries can be used to manipulate the objects, humans, fasteners, and tools in the DEPTH environment. The standard motions can, in turn, be sequentially combined to build a complex maintenance task that is associated with each task element. Frames from the simulation, as well as movies can be captured for later transfer to the LSAR database.

Facilities are provided to debug the task steps. Each motion is added to the simulation as a motion step and is visually executed as the simulation is built. Erroneous motions can be detected immediately and removed from the simulation. A new step can then be added to correct the problem. Finally, the result can be saved in a simulation file that can be reloaded and run at a later time.

When the simulation is deemed satisfactory, EAGLE is updated by running the simulation file using the appropriate simulation menu. When a simulation is run to update EAGLE, the database is queried and updated interactively using SQL commands. At the end of the run, the appropriate fields in the database will have been updated with information from the DEPTH simulation.

3 Conclusions and Recommendations

It is technically feasible for DEPTH to produce text describing maintenance task steps suitable for IETMs. The language initially produced will consist of a simple instructions – an action verb followed by a target. However, this should suffice to meet the requirements of IETMs.

It is also technically feasible for DEPTH to produce graphical illustrations and suitable movie inputs for an IETM. As with HTML used on the World Wide Web, a link can be created to any type of file including graphics, animations and Virtual Reality Modeling Language (VRML).

Tasks that involve motion are the best candidates for DEPTH given the graphical nature of the simulations. Steps involving cognition or logic (analyzing, deciding, comparing, etc.) are not as well suited without significant additions to the system.

With some minor modifications, DEPTH can produce text for warnings (and cautions) associated with maintenance. However, there is currently no substitute for human judgment of this area. Searching the environment and generating warnings for all objects is too involved for current simulation systems. Since DEPTH will generate no warnings unless a hazard region is contacted, human judgment is needed to author additional warnings.

Text generation of task steps should be kept within the simulation environment. The Transom Jack PaT-Net level of text generation proposed by the University of Pennsylvania is too low level. It is too difficult to decide automatically which detailed motions can be incorporated into and referenced with a higher level motion.

DEPTH should use commercial IETM authoring tools rather than duplicating that functionality. DEPTH can create the data (text, motion sequences, screen capture, movies, and supporting information) that is input by a human to an authoring tool or database. The problem is the bottleneck of information between the human and the system. This can be DEPTH's strength. Creating SGML tagged output for display of an IETM is one the strengths of commercial IETM tools.

4 Bibliography.

Advanced Integrated Maintenance Support System (AIMSS) Author's Guide. Hughes Missile Systems Company.

EAGLE Workbook. Hughes Missile Systems Company.

Mann, T.L. 1997. "Storyboards and Design Notes for DEPTH LSAR Database Interface".

MIL-STD-1388-2B

R. Vujosevic, R. Raskar, N. V. Yetukuri, M. C. Jothishankar, and S. H. Juang. Simulation, animation, and analysis of design disassembly for maintainability analysis. Int. J. Prod Res., 1995, Vol. 33, No. 11, 2999-3022.

5 Appendix A: Natural Language Generation From Task Networks For Technical Manuals

Natural Language Generation

From Task Networks

For Technical Manuals

Norm Badler, Bonnie Webber, Martha Palmer, Tsukasa Noma, Matthew Stone, Joseph Rosenzweig, Sonu Chopra, Ken Stanley, Hoa Dang, Rama Bindiganavale, Diane Chi, Juliet Bourne, and Barbara Di Eugenio

University of Pennsylvania

March 31, 1997

Table of Contents

LIST OF FIGURES

List of Charts

1. Executive Summary

In this appendix, parameterized action representations (PARs) are introduced. PARs comprise an intermediate level of representation between PaT-Nets (task networking language embedded in Jack) and natural language instructions. The PAR is an explicit structure representing actions that are constructed either from an appropriate menu-driven user interface or through some future language understanding tool. PaT-Nets are the programming language from which simulations are produced and hence correspond to an execution-level implementation of the PARs. PARs are therefore being structured with the view that "run-time" process can interpret sets of PARs into PaT-Nets and thence simulate virtual humans.

Conversely, by their derivation from the syntax and semantics of English instruction words, PARs offer an appropriate platform for the generation of natural language descriptions of the actions they embody. The PARs include information about the participants – the agent and the object(s) – as well as applicability conditions and culminating conditions. Relevant spatiotemporal and manner information and is also included, as well as specification of subtasks. This can provide the details necessary to characterize the salient information.

We have described the three separate stages of natural language generation as text planning, sentence planning and surface realization. There are currently two major approaches to text planning, which ensures the coherence of the multi-sentence text that is being generated: the schema approach and the planning operator approach. The schema approach, which is based on pre-determined patterns of information, is preferable for situations such as technical order generation, where the emphasis is on rapid implementation and efficient execution. In other situations where it is critical to add additional detail or answer follow-up questions, the planning operator approach which reasons explicitly about intentions would be preferable even though it is harder to implement and validate than the schema approach.

In applying the schema approach to our technical order generation domain, we have discovered the need for two types of schema, one for procedural descriptions, and one for warnings and cautions. The procedural description schema consists of action specifications which often include conditions for initiation or termination of the action, as well as conditions for performance such as necessary coordination with other actions. The purpose is also often described. Examples of the types of procedures that might be represented as schemas include installations, removals, connections and disengagements.

The schemas for warnings and cautions precede the procedural descriptions to which they apply, and include specifications about potential hazards and necessary precautions that should be taken. They often include potential results of disregarding precautions. Example verbs that are frequently used include "exceed," "check," "make sure," and these expressions are often annotated with "when" clauses.

The PARs that correspond to the actions being performed in the simulation trigger the generation of one or more schemas capable of structuring the description of the procedure being carried out. It is then necessary to link the culminating conditions in the PAR, if explicit, to culminating conditions in the schema, as well as the purpose, etc. The next step is to plan the sentences that will describe the actions that instantiate the schema. This may require breaking an action into subtasks so that the participants and results can each be described at the appropriate level. Choices about noun phrase content will be made depending on the context, and whether or not the objects involved have already been completely specified. If they are being introduced for the first time, it might be necessary to include helpful properties such as "the receptacle mounting lug, located next to X." If they have just been mentioned in the previous sentence, then a pronoun such as "it" might be sufficient. The verbs and their arguments must also be chosen, and this will require reference to object-specific information about the objects involved and typical processes they participate in. For instance, the PARs and associated descriptions for removing caps and nozzles depend on the type of object being removed and its typical home position, even though the end results are the same: that the object is not longer in the home position. Removing a cap involves a rotation action, and removing a nozzle requires a lifting action along the filler tube path. The different actions are stored within the object instance or class.

```
remove(cap)         -> grasp(cap)
rotate(cap) - until loose(cap)
move(cap, Location Q)
release(cap)

remove(nozzle)           -> grasp(nozzle)
lift_up(nozzle)
```

Other actions, such as a push action, may be much more independent of the type of object, and a more general, generic PAR can be applied to entire classes of objects with any necessary modifications being made dynamically.

Having planned the text and the sentence, it remains to produce the surface realization, ensuring that syntactic and semantic features unify and that the resulting text is grammatical.

2. Instructions

A virtual human simulation must represent human physical capabilities and limitations. For cognitive and intellectual domains, the computer must understand reasoning, decision-making, and communication. One connection between these two domains lies in the understanding and execution of instructions: commands or descriptions of physical activities or their consequences. Accordingly, the computational representation of information and processes sufficient for instruction understanding (from textual material into action) and text generation (from task performance simulation) is a challenging research topic.

This study examines the feasibility of bridging the computational gap between the simulations of the physical world and the symbolic world of language as instructions for human activities in those environments. Our scope is somewhat further limited in this document to examining text generation rather than understanding, though we have current and historical interest in the instruction understanding problem as well [BWKE91,BPW93,WBE95].

Textual instruction generation is well motivated by the desire to partially automate the production of Technical Orders (instruction manuals) for Air Force maintenance and repair activities. The desire to engage computational tools in this process is motivated by several unique characteristics of Technical Order (TO) production:

- TOs must be accurate, understandable, and executable across a broad range of repair personnel.
- Military system complexity, customization, and design evolution creates significant TO update requirements.
- The volume of TOs for a complex military vehicle or platform, such as a tactical fighter, demands computational tools to reduce the sheer logistics of physical materials and well as enhance the delivery of the information through computer screens.

Increasing use of digital mockup and virtual prototyping tools is providing digital environments in which human tasks may be tried and tested. Graphical systems such as DEPTH and Jack [BPW93] are making design evaluation and human factors analysis feasible prior to system production and deployment. At this stage, it is desirable to debug maintenance tasks and correct errors in the design itself. Once the maintenance actions appear feasible (relative to the expected population of maintainers), the analyses should be preserved, and in fact a written record of the analysis is all that is required now (by storing a record in an LSAR database). An important observation is that additional useful information is potentially available that can aid in the generation of the requisite TO: namely, the simulation "commands" that created the task analysis may be suitable as a framework for TO textual descriptions of that same task. This opportunity requires careful study, though, as the expression of human actions for simulation or analysis is not the same as that used for TOs. While a desirable goal, current knowledge is simply not yet ready to deliver that kind of performance.

Our goal in this document to examine the extent of the gap between simulation and instruction generation, and make recommendations for research and development efforts that have the potential to reduce or eventually close this gap. Accordingly, this report focuses on two main themes. The first concerns the representational issues surrounding computer models of processes and human actions in simulated environments. The major goals of this part are to try to design a concept-rich structure that will capture characteristics of tasks in terms of sensing, acting, and decision-making, to justify the computational feasibility of the representation, and to show potential influences will be exerted on the task simulation user interface. The second theme centers on Natural Language text generation. The major emphases are on methodologies of text generation, specialized requirements for TOs, the role of verbs and objects in existing TOs, and examples from the maintenance domain.

3. Process Representations for Analysis, Synthesis, and Description

The underlying thesis of this section is that any effective communication across the language and action chasm requires an intermediate representation language that supports concepts from both. Fortunately we have had many years of experience in building computational models for both, and in fact, have deliberately aimed toward representations that facilitate connections.

One way to view the requirements for such a common representation is to observe the situations in which information in one input modality (motion, text) is to be converted into information in a different output modality (text, animation). Ideally, the representation would support all such transformations:

- From 3D motion capture data to graphical animation (the current approach to performance animation);
- From 2D visual (video) motion capture data to graphical animation (the computer vision motion understanding process);
- From Natural Language instructions to graphical animation (our long-standing AnimNL project);
- From a programmable view of the representation into graphical animation (for example, programming animation with PaT-Nets (Parallel Transition Networks) through VisualJack or OMAR [BBN94]);
- From the programmable view of the representation into movement descriptions (converting PaT-Nets into text-based instructions).
- From a simple command-based language syntax into graphical animation (the UPenn and Lockheed-Martin collaboration on JackMOO, a multi-user, real-time, shared textual environment augmented with 3D Jack avatars).

Rather than approaching each of these as a separate problem, we build an alternative theory based on a process representation which admits and facilitates all of them. These interrelationships are diagrammed in Figure 1. The main tenets of this theory are:

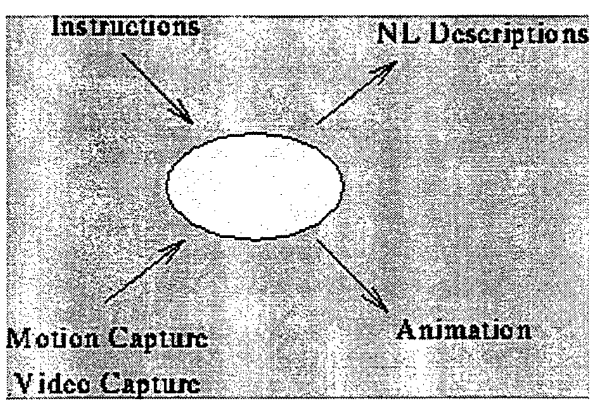


Figure 1: Using process representations

- Representing processes and their circumstantial, causal and intentional relations with states and other processes over time, is the core idea in the conversion of the various media.
- Process representations must function as recognizers, predictors, and descriptors.

- Designing the process representation with this broad scope will prevent the design or use of arbitrary (un-constrained) structures, i.e., people who design/build processes that generate output (text or animation) must adhere to structure conventions; user interfaces may help or force the designer to build only "correct" structures.

- PaT-Nets correspond to the execution-level implementation of a process representation.

- PaT-Nets require semantic definitions and suitable restrictions to permit them to be compiled from a process representation.

- The structure of language (both descriptions and instructions) provides motivation for the process concept vocabulary and structure.

- The kinematics and dynamics of motion and change provides motivation for the process internal definition and facilitates conversions across modalities.

- Process representations are hierarchical; the lowest levels ground out in performable (executable) actions, middle levels coarticulate (blend) and arbitrate among parallel or competing actions, and higher levels agglomerate these recursively into meaningful action, task, or conceptual units.

- Objects and mechanisms have structure, attributes, and processes which can be modeled by various means -- including input-output "black-boxes," external simulations, physics-based simulation, etc.

- Human-like agents perform actions with physical (movements, manipulations), cognitive (think time), and sensing (attention, observation, testing) requirements..

- Agents have individualized, limited (e.g. two hands, one eye gaze direction), and variable (e.g. strength, fatigue, reach time, reaction time) resources.

- Agents have skill levels, roles and responsibilities that may affect how/whether they can be bound to specific processes or higher-level actions.

- Process representations require parallelism and coordination among the various objects and agents in the environment, and among the resources of any particular agent.

- Sensing is an essential part of representing an agent's actions in the world, and sensing takes time, repetition, and resources.

- Cautions and warnings alert the agent to particular sensing and acting requirements relevant to the task.

While there are many related topics of interest, we will put them aside for now. In particular, we will not address:

- Inferring intentions or causality beyond that which follows from PaT-Net structures or explicit annotations thereof.

- The issue of learning a process representation by being shown examples of either motions or textual instructions.

- The understanding of free Natural Language text, whether in instructions or not.

- The role of chance or interruptions in activity caused by unexpected or unpredictable events except as necessary to capture the essence of a particular action.

Section 4 fully describes the proposed process representation as well as the agent and object representation specifics in Agent (Section 4.1.1) and Objects (Section 4.1.2).

4. Processes and Actions

Processes are :

1. A system of operations in the production of something.
2. A series of actions, changes, or functions that bring about an end or result.
3. Course or passage of time.
4. Ongoing movement; progression.

So processes have two general forms: one in which something is happening and another in which something is completed. The conventional terminology is that the latter are culminated processes. We will also term any non-culminated process active to clearly distinguish this case. It is therefore convenient to graphically present processes as nodes in which some "action, change, or function" takes place, and arcs which link one process (node) to another that temporally follows either by virtue of culmination of the first or other circumstances. A process can be recursively defined as a network (or graph) of process nodes (possibly disconnected, i.e. parallel). Thus, a hierarchy of processes can exist, grounding out at single process nodes for the simplest types of processes.

An action is just a particular kind of process which involves a volitional agent acting in the world. We call our representations of actions Parameterized Action Representations (PARs) and they contain a necessary slot for an agent. A generic process representation is a PAR with an optional agent slot. In this report, we deal almost exclusively with PARs that contain the necessary agent slot.

Our representation is a modified version of the representation used by Kalita and Lee [KL96], expanded to include culmination conditions, agent/object representations, as well as more detail about the specifics of actions.

4.1 Parameterized Action Representation Specification

The top-level type in the representation is the parameterized action (see Figure 2); we call it "parameterized" because an action depends on its participants (agent and objects) for the details of how it to be accomplished. For instance, opening a door and opening a window will involve very different behaviors on the part of the agent. The subparts of a parameterized action can refer to particular aspects of the agent and objects as part of their meaning. In the sections that follow, we specify and discuss each subpart of the representation.

```
type parameterized action =
        (agent: agent representation;
         objects: sequence object representation;
         applicability conditions: sequence disjunctive-queries;
         culmination conditions: sequence disjunctive-sensor-queries;
         spatiotemporal: spatiotemporal specification;
         manner: manner specification;
         subactions: actions).
```

Figure 2: The parameterized action type

4.1.1 Agent

As mentioned above, the agent is the distinguishing feature between an action and a mere process. It specifies which agent is carrying out the process described in the rest of the representation. We assume that the agent refers to a human model or a physical force like gravity (in which case the agent is understood to be causal and not volitional).

The agent type (see Figure 3) and the object type (see Figure 4 in Section 4.1.2) represent agents and objects respectively. They are very similar in concept except that the agent type has some extra fields which also describe the behaviors of the agent which would influence some of the actions of the agent.

For each instance of the agent type, a list of actions that the agent is capable of performing is specified. The agents can also be considered to be capable of playing different roles. For each role, the agent performs different actions. So, instead of maintaining one long list of actions, we could group these actions under different roles. For example, the actions involved while driving a car like grasp a steering wheel, sit with foot on the accelerator pedal, etc., would be grouped under the "car-driver" role. Each of the listed actions is a primitive action. Unlike for the objects, each action is associated with a set of applicability conditions (test for accessibility, etc) which check if the action can be performed by the agent. If not, another set of primitive actions is generated for that agent which have to be completed before the current action can be performed. The agent type also has a field for specifying nominal values and the distribution type and range for some of the actions and state space descriptors. For example, the walking rate of the agent could be specified to have a nominal value of 2 steps per second with a normal distribution and standard deviation of 1. This gives a range of values over which the walking rate can be varied.

```
type agent representation =
        (coordinate-system: site;
         state: state space;
         rel-dir: relative directions;
         special-dir: special directions;
         grasp-sites: sequence site;
         capabilities: sequence actions-and-applicability;
         nominals: sequence value-ranges).

type actions-and-applicability =
        (action: parameterized action;
         applicability: sequence disjunctive-queries).

type value-ranges =
        (var: powerset parameter;
         mean: powerset var-types;
         standard deviation: powerset var-types;
         distribution: powerset distribution).

type parameter =
        (id: string).

type var-types =
        (real, real vector, integer).

type distribution =
        (normal, poisson, uniform).
```

Figure 3: The agent representation type

4.1.2 Objects

The object type is defined explicitly for a complete representation of a physical object (see Figure 4). Each object in the environment is an instance of this type. We could also distinguish between objects and causal models.

The state field of an object describes a set of constraints on the object which leave it in a default state. The object continues in this state until a new set of constraints are imposed on the object by an action which causes

a change in state. The other important fields are the reference coordinate frame, a list of grasp sites and directions defined with respect to the object.

For each instance of the object type, a set of actions are defined. Each of these actions can be further described as a group of one or more actions. Also, the objects can be represented hierarchically. This allows us to describe actions for a class of objects rather than for every object. The actions are defined at the highest possible level in the object tree. So the action field in an instance of the object type could point to a description or to the parent.

```
type object representation =
        (coordinate-system: site;
         state: state space;
         rel-dir: sequence relative direction;
         special-dir: sequence special direction;
         grasp-sites: sequence site;
         actions: sequence parameterized action).

type site =
        (position: real vector;
         orientation: real vector).

type state space =
        (position: real vector;
         velocity: real vector;
         acceleration: real vector;
         force: real vector;
         torque: real vector).

type relative direction =
        (name: (front, back, left, along, inside);
         value: real vector).

type special direction =
        (name: string; value: real vector).
```

Figure 4: The object representation type

4.1.3 Applicability Conditions

The applicability conditions of an action specify what needs to be true in the world in order to carry out an action. These can refer to agent capabilities, object configurations, etc., and are represented by conjunctions of disjunctions of queries on the state of objects, agents, and global variables (see Figure 5). For instance, the applicability conditions for changing a light-bulb could be represented as: has(agent, light-bulb) AND (can_reach(agent,light-fixture) OR has_ladder(agent)).

```
type disjunctive-queries =
        (query, sequence query).

type query =
        (object-query, agent-query, global-query).

type object-query =
        (obj : object; oquery: string).

type agent-query =
        (agent: agent description; aquery: string).

type global-query =
        (variable: global variable; gquery: string).
```

Figure 5: Applicability conditions as a series of disjunctive-queries

4.1.4 Culmination Conditions

Whether an action is considered a process or a culminated process depends on whether a culmination condition (default or otherwise) is specified. Processes do not have "culminations" associated with them; that is, processes just end, with no consequences attached to their ending. Culminated processes, on the other hand, can only be said to have occurred or be completed if they have reached their culmination, in which case the consequences are that the culmination conditions hold.

```
type disjunctive-sensor-queries =
        (sensor-query, sequence sensor-query).

type sensor-query =
        (agent: agent description; squery: string).
```

Figure 6: Culmination conditions as a series of disjunctive-sensor-queries

Culmination conditions, or the conditions that will hold when an action is completed, are also a series of queries like applicability conditions, however they are restricted to queries of the agent's sensors (see Figure 6). That is, the only way an agent is allowed to determine information about the world is through its sensors. For instance, an agent cannot simply query an object, say the lid of a jar, to see if it loose; the agent must query a sensor to find out if the lid is loose. So, instead of a query such as loose(lid), the agent must use a query such as sense(loose(lid)) in order to determine whether the culmination condition in the action "Turn lid until loose" holds. Such sensing processes are needed independently of the need to represent culmination conditions (see Section 4.2.2).

4.1.5 Spatiotemporal Specification

At an abstract level, basic actions involve manipulating spatial or geometric relations between objects (or possibly between the agent and objects). Thus, part of the spatiotemporal specification of an action can be a spatial goal, which consists of the type of the manipulation (establishing, terminating, maintaining, or modifying) and the relation that is being manipulated (see Figure 7). The relation is expressed as queries, which will refer to spatial predicates on objects and agents. For instance, the action in "Put the block on the table" would have the spatial goal of establishing the relation of the block being on the table. The relation could be expressed as a query to the block, "block.on(table)", which would return a "yes" or "no" answer. The relation would be considered established when the query returned a "yes." In order to convert these abstract spatial goals into the lower level kinematics and dynamics, an algorithm is needed that can understand the

queries (e.g., what it means for something to be "on" something else) and convert them into motions for establishing, terminating, etc., the relations they express.

```
type spatiotemporal specification =
        (goal: spatial-goal;
         kinematics: kinematics specification;
         dynamics: dynamics specification;
         frequency: rate specification).

type spatial-goal =
        (execution-type: (establish, terminate, maintain, modify);
         relation: sequence disjunctive-queries).

type kinematics specification =
        (position: real vector;
         time: time specification;
         velocity: real vector;
         acceleration: real vector;
         path: path specification).

type dynamics specification =
        (force: real vector;
         torque: real vector).

type rate specification =
        (period: real vector;
         amplitude: real vector).

type path specification =
        (direction: (relative direction, special direction, real vector);
         scale: real;
         pathpoints: sequence 3Dpoints).
```

Figure 7: The spatiotemporal specification type

The spatial goals are provided by the lexical terms used in the input or, conversely, are formed by recognizing certain changes in directions or relationships in the object state. In either case we consider a set of terms including those considered and defined in Badler's earlier work [Bad75]:

across	after	against	ahead-of	along
apart	around	away	away-from	back
back-and-forth	backward	behind	by	clockwise
counterclockwise	down	downward	forward	from
in	in-the-direction-of	into	inward	off
off-of	on	onto	onward	out
out-of	outward	over	sideways	through
to	together	toward	under	up
up-and-down	upward	with		

The semantics of these terms for animation synthesis is to be implemented through PaT-Nets that transform the given parameters (e.g., objects involved) into movement paths, vectors, or directions eventually processed by the primitive actions sent to the agent. For textual synthesis, PaT-Net "recognizers" as defined in [Bad75] can

be executed to vote for the semantic term most closely describing the changing situation being presented. Although many terms are simply recognized
from directions relative to object labels ("forward" from movement with front of object in the lead, etc.), others require a multi-node PaT-Net to determine the sub-steps and completion of the activity (e.g. across, around, back-and-forth, etc.)

Mathematically, a motion can be represented in any one of the three domains - kinematic, dynamic and frequency. The motion generating primitive functions take parameterized inputs to generate the exact motions. The modifiers impose additional constraints on the motion. At any time, one or more of the constraints are active. The constraints can be specified in any of the above mentioned three domains. The mathematical components of the representation are designed to facilitate conversion of movements defined or specified in one system. Conversion procedures exist to change kinematics data into dynamics information (inverse dynamics) and vice versa (forward dynamics) [KMB96].

The basic parameters for modifying a motion in the kinematic domain are position, time, velocity, acceleration and path. These parameters are in general relative but they could also be global. In the dynamic domain, given a motion and the agent/object involved, it is possible to compute the forces and torques at the joints required to generate the given motion. If an object is able to rotate about an axis, then a force applied in a direction perpendicular to the axis of rotation causes the object to rotate. The speed of rotation is governed by the magnitude of the force. So, to modify the motion in the dynamic domain, we can specify relative forces and torques. In the frequency domain, we can represent the motion as a function of time using Fourier series expansion. The parameters to vary the motion in this domain could then be mapped to period and amplitude.

4.1.6 Manner Specification

At the lexical level is an enumeration of manner adverbs related to verbs. At the action representation level, we have the "manner" or "Effort-Shape" (ES) parameters described below. To go from the lexical level to the action representation level, we would have procedures that operate on the (verb, adverb) pairs to translate them into manner-affecting ES parameters. One set of adverbs will modify the ES parameters directly, taking the default values from the stored nominal values. So an instruction to walk quickly would involve looking up the nominal walk speed and invoking a generic procedure "quicker" that scales the nominal rate by a factor of say 50% toward the maximum walking rate. This scaled rate can now be filled in the agent's current forward motion rate and used by the locomote action. A second class of adverbs could be interpreted as members of the first with some assumptions. For example, "carefully" can be interpreted as slowly and with low accelerations (something that can be consumed by the ES notation). In some cases, then, an adverb tells what movement pattern to apply to a (novel) situation. The patterns would be (pre-stored) action (fragments) that would be newly bound to the current object and agent. So these adverbs are interpreted by procedures that take the action as a parameter, substitute new object bindings, and then compose the spatiotemporal characteristics into the current action being done.

Manner specifications describe the way in which an agent carries out an action. We define manner as composed of an Effort parameter and a Shape parameter (see Figure 8). The Effort parameters, which are derived from Effort Notation [BL80], express the quality of a movement. Each parameter takes on a real value in the range from -1 to 1. The Effort elements include Space, Weight, Time, and Flow. Space varies from direct (1) to indirect (-1) and describes whether a person is focusing in on his/her surrounding space or not. Weight expresses the forcefulness of a movement at its end, ranging from light (-1) to strong (1). The Time element describes the sense of urgency in the beginning of movement along the continuum between sustained (low acceleration) (-1) and sudden (high acceleration) (1). Flow describes the progression of a movement and ranges between free (dynamically-driven) (-1) and bound (kinematically-driven) (1). The Shape specification describes the general shape and pose of the agent's body, giving information on the directional goals of the agent. The Shape parameters are specified for each plane -- vertical, lateral, and sagittal -- as real values in the range [-1,1]. The vertical parameter varies from sinking (-1) to rising (1), lateral varies from narrowing (-1) to widening (1), and sagittal from retreating (-1) to advancing (1).

```
type manner specification =
        (effort: effort parameters;
         shape: shape parameters).

type effort parameters =
        (space: real range[-1,1];        /* Indirect = -1, Direct = 1 */
         weight: real range[-1,1];       /* Light = -1, Strong = 1 */
         time: real range[-1,1];         /* Sustained = -1, Sudden = 1 */
         flow: real range[-1,1]).        /* Free = -1, Bound = 1 */

type shape parameters =
        (vertical: real range[-1,1];     /* Sinking = -1, Rising = 1 */
         lateral: real range[-1,1];      /* Narrowing = -1, Widening = 1 */
         sagittal: real range[-1,1]).    /* Retreating = -1, Advancing = 1 */
```

Figure 8: The manner specification type

4.1.7 Subtasks

Complex actions, such as fueling a vehicle, will involve several subtasks. Thus, the subtasks part of the representation consists of a list of parameterized actions and temporal constraints among the actions (see Figure 9Figure 9). The possible temporal relationships between two actions are:

```
type actions =
        (actions: sequence labelled actions;
         constraints: sequence constraint-and-labels).

type labelled actions =
        (label: string; action: parameterized action).

type constraint-and-labels =
        (constraint: (seq, par, par-join, par-indy, while);
         action-label1: string; action-label2: string).
```

Figure 9: The subtasks type

Sequential (seq)	The actions are done sequentially (default)
Parallel (par)	The actions are done in parallel
Jointly parallel (par-join)	The actions are done in parallel and no other actions are done until after both have finished
Independently parallel (par-indy)	The actions are done in parallel but once one of the actions is finished, the other one is stopped
While parallel (while)	The subordinate action is done while the dominant action is done; once the dominant action finishes, the subordinate action is stopped

A complex action can be considered done if all of its subtasks are done or if its explicit culmination conditions are satisfied.

4.2 Action Representation to PaT-Nets Control Algorithm

In order to produce simulation, actions represented in the manner described above must be converted into PaT-Nets. All the actions of an agent which correspond to a given set of instructions are referred to as the top-level actions and are maintained at the highest level in a queue tree. Each of these high level actions might have subtasks. All these subtasks are now maintained in a queue at the next level. For every action, a PaT-Net is

spawned. For every high level action, the subtasks form the children and the higher level action is assumed completed only after all the children's actions are completed. An action is also considered completed if the culmination conditions of some higher level PaT-Net are satisfied. A sequence of actions is maintained as children from left to right, the leftmost child being executed first. Once an action is completed, the action on its right is then considered. However, if any set of actions have to be executed in parallel, we use the concepts of par-join, par-indy, etc (see Section 4.1.7).

Given an action specification from the top-level queue, there are three possibilities:

1. The action is a primitive action and is added to a lower level queue. Once there, the applicability conditions for the action are checked. If the conditions are true, then a PaT-Net that corresponds to the primitive action is spawned for the execution of the action. If the conditions do not hold, another list of subtasks may be added as children of the current action node in the queue tree. (See Section 4.2.1 for more on primitive actions.)
2. Object-specific information, such as an action definition defined specifically for an object involved in the action or directions relative to an object, etc, needs to be obtained. The expansion of the action with object-specific information is added to the queue tree below the current action node, including any subtasks. (Whenever the object or agent is accessed for information on the action, it is first checked if it has been defined for that agent/object, possibly higher in the object hierarchy. If not, then an error message is generated.)
3. The action can be expanded into subtasks. Each of these subtasks then forms a child of the current action node in the queue hierarchy.

See Figure 10 for a simplified version of the control algorithm which ignores most of the details of the queue hierarchy that is maintained.

```
For each parameterized action, Act, in the Instruction-queue
    Initialize Q to be an empty queue
    If Act.subactions is non-empty then
        Put each element in Act.subactions into Q
    Else put Act into Q
    While Q is not empty
        Let A be the element at the head of the queue
        If A is a primitive action then
            Let P be the PaT-Net corresponding to A
            Put P into the Primitive-queue
        Else if A needs object-specific expansion then
            Let L be the result of object-specific expansion on A
            Put each element of L into Q
        Else if A.subactions is non-empty then
            Put each element of A.subactions into Q
```

Figure 10: Action Representation to PaT-Nets Control Algorithm

4.2.1 Primitive Actions

Currently, we refer to the basic actions in Jack as the primitive actions. These include actions such as move (which includes translate and rotate within it), grasp, reach, locomotion, visual search, task-appropriate attention, etc. Each of these primitive actions can be generated by various methods like motion scripting, live motion capture, various algorithms using the principles of dynamics and control, fast inverse-kinematics, random noise, walking algorithms, etc. Instead of generating motions for the whole agent (we consider only the human agent for this purpose) using only one of these sources at a time, we could use a different source for each part of the body. For this purpose, currently, at a higher level, the body of the human model is divided into the following parts - head, arms, torso, legs, and figure position. A different PaT-Net is used for each

motion generator. The parameters to the motion generator would then be the part of the body that they would control. If more than one motion generator were to control the same part of the body, it could be resolved in two ways - blend (or any other binary operation) the resulting motions together or set up a priority for the motion generators and arbitrate appropriately between them.

4.2.2 Sensing processes

An important contribution of the Jack virtual human model is that it allows us to view and analyze human actions such as locomotion, reach, grasp and gesture. Tasks composed of such elementary actions may in turn be combined and performance observed. Simply stringing together sequences of tasks, however, is unrealistic since humans require time to visually attend and acquire information before or during execution of tasks. The action representation requires a framework of sensing processes in which visual search, hand-eye coordination and attention guides the sequencing and simulation of actions, especially motor tasks. Other sensing activities that we must represent are monitoring, probing the state of the world via global or internal variables, checking time on global or relative clocks and evaluating spatial/geometric predicates (relationships).

We contend that sensing activities or processes are themselves tasks whose duration and execution are significant. Further, timing information for processes such as attention and visual search is not static. Such information will vary with cognitive load, expertise and type of elementary action. Studies show that eye movement latencies are longer when accompanied by certain spatially oriented gestures like pointing[BKA95].

The Jack system maintains a geometry of objects and relevant sites in a virtual world. Since access to these objects is negligible in terms of computer simulation time, some explicit representation of sensing or investigative process is particularly important. Further efforts are required to model the time durations introduced by sensing and attentive processes. Part of this problem is being addressed by explicit attentional models constructed within the OMAR [BBN94] system. Since it is likely that a comprehensive attentional timing model would be difficult to develop, we may first incorporate simplified reaction time computations to achieve a modest level of performance veracity.

The sensing processes themselves include a number of particular features:

- Spawn, if necessary, a sub-net with attentional and/or manipulative and/or cognitive actions.
- Check time on global or relative clocks.
- Send messages to probe external global variables.
- Send messages to probe specific internal states of other processes.
- Evaluate spatial/geometric predicates (pseudo-pattern recognition for relationships, situational awareness, etc.).
- Include sampling frequency (independent of simulation clock ticks).

To the extent made possible by the functional definitions, the future variable values of a process node may be queried by a sensing process. In the case of a formula for linear motion, for example, the present trajectory may be extrapolated to yield an estimate of possible future location. The internal object description includes fields for velocity and acceleration as well as position and orientation. It is speculated that the ability to successfully run processes in a predictive mode will improve agent/agent and agent/object interactions by changing "pursuit" situations into "anticipation" situations [Rei97].

4.2.3 Queue of Primitive Actions and Visual Attention

Requests for primitive actions are placed on a queue. A queue manager is maintained that consumes such requests and determines which agent resource should be used to execute the request. For example, the queue manager may determine which hand should be used to execute a grasp based on following criteria: which hand is empty, which hand is closer to the object to be grasped and also potentially which hand the agent favors for object manipulation (e.g., is the agent right or left handed?).

A queue of action requests facilitates the simulation of sensory procedures. Such procedures may be executed parallel or in addition to motor actions. For example, task related attention is supported in our human model. The queue manager automatically invokes the appropriate attentional behavior for each type of task on the queue. While the agent is walking to a goal, for example, he will look at the goal and occasionally glance at his feet(when a memory uncertainty threshold is exceeded or when the agent is in close proximity to an obstacle). When grasping an object, the queue manager invokes an attentional process that determines which sites are relevant for the grasp and directs attention appropriately. Our aim is to direct attention as autonomously as possible and based on analysis of task mix. Since a queue allows look-ahead of downstream tasks, we allow the potential to interleave or anticipate where attention may be directed.

5. Action Representation Issues for Natural Language and Technical Orders

In designing the parameterized action representation (PAR), several issues from the natural language perspective guided our choices. The action representation scheme is an intermediate representation language between the simulation constructs, i.e. PaT-Nets, and natural language instructions. It allows a hierarchy of actions to be represented explicitly (by listing subtasks for an action). This hierarchy is needed in order to generate natural language at the correct description level. PaT-Nets can also form a hierarchy of structures. In fact, PAR shares enough in common with PaT-Nets that the conversion between PAR and PaT-Nets is relatively straight-forward. However, one aspect in which they significantly differ, in terms of representing actions, is that PaT-Nets require instantaneous transitions from one node in a network to another whereas the corresponding concept in our representation, i.e. culmination conditions, does not have this restriction. Culmination conditions are represented as queries to sensors, which as described in Section 4.2.2 can have significant duration, even to the point of having subtasks of their own. Thus, the culmination conditions in our representation might be translated into PaT-Nets that need to be executed in order to check whether the culmination conditions hold. In generating instructions, we will not usually generate text corresponding to the internal structure of the sensing process, so we do not need it in our action representation and can leave the details of expanding the sensor queries to a lower level.

The overall picture of how PaT-Nets, our action representation, and natural language all fit together is shown in Figure 11. The control algorithms for the (a) and (d) transitions are described in Section 4.2 and Figure 13 in Section 7.2, respectively. (The (b) and (c) transitions are not currently relevant, but they are certainly kept in mind.)

```
PaT-Nets <—> Action Representation <--> NL Instructions
        a b                        c d
```

Figure 11: The action representation as an intermediary between PaT-Nets and natural language

Task simulations from which to generate natural language will be restricted to be in PAR form by providing an authoring system that enforces the specification requirements of the action representation. Through the conversion of PARs to simulations, task simulations can be performed while preserving the structure that is suitable for generating natural language.

6. Text Generation

6.1 Overview

In existing natural language generation systems, the process of text generation is divided into three main stages: text planning, sentence planning, and surface realization [Rei94], as in Figure 12.

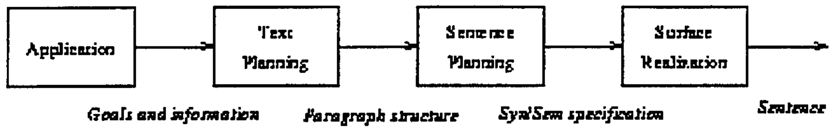


Figure 12: The stages of language generation

This section briefly describes these tasks, and the issues that arise in performing them in describing the execution of processes. The key point is that to produce the kinds of clear and understandable texts that people need requires rich models of both the causal relationships in the domain, and the inferential capacities of the reader or hearer to reason about those relationships. Although this information must be given explicitly to a natural language generation system, it generally consists of "obvious" commonsense facts, and is accordingly painstaking and uninteresting for users to construct. We therefore outline some research strategies for allowing Pat-net designers to specify needed knowledge as naturally and as unobtrusively as possible during programming.

6.1.1 Text planning

Natural language systems need to generate not merely text, but coherent text. It is not enough to decide on a collection of facts and string a description of those facts together. The facts must be organized so as to signal the causal, logical and intentional relationships between them. Often, these relationships should even be explicitly indicated in the text, using special connectives [Hov88] provides a convincing example of the importance of presenting information linked together in the right order and with the right connectives. He observes that the following discourse is difficult to understand:

> The system performs the enhancement. Before that, the system resolves conflicts. First, the system asks the user to tell it the characteristic of the program to be enhanced. The system applies transformations to the program. It confirms the enhancement with the user. It scans the program in order to find opportunities to apply transformations to the program.

Meanwhile this discourse, which conveys exactly the same propositions, is relatively clear:

> The system asks the user to tell it the characteristic of the program to be enhanced. Then the system applies transformations to the program. In particular, the system scans the program in order to find opportunities to apply transformations to the program. Then the system resolves conflicts. It confirms the enhancement with the user. Finally, it performs the enhancement.

Researchers have argued that the difference between examples such as these arises because natural discourses have an intentional structure [GS86]. Discourses can be broken up into nested blocks of contiguous material, called segments, on the basis of how the material contributes to the speaker (or writer)'s plans for presenting information. Each segment is associated with a discourse segment purpose which describes this contribution and which the speaker expects the hearer to recognize as part of understanding the discourse. Cue words, like finally or in particular above, facilitate this recognition by making explicit the argumentative relationships

between segments---so finally marks the concluding step in an argument, and in particular introduces a more detailed description of a previously mentioned process or generalization [Coh87].

There are two somewhat competing approaches, schema- and plan-based, that bring this idea to bear in natural language generation. The earlier work on text planning, which [McK85] pioneered, used schemata: schemata represent naturally occurring patterns of discourse. For example, McKeown's system constituted the interface to a database containing knowledge about different kinds of ships. Thus, one of her schemata to describe a concept includes instructions to identify its superclass, to name its parts, and to list its attributes. Schemata in McKeown's TEXT system were implemented by means of ATN's: TEXT traverses the schemata and sequentially instantiates rhetorical predicates with propositions from the knowledge base. Subsequent work (in particular [MP93,Caw92,Moo95]) pointed out that a major problem with schema-based text planners is that they cannot reason about the structure, content and goals of explanations. This is a very important issue for systems that must be able to answer follow-up questions, or to replan how a certain goal has been achieved, in case the user is not satisfied with or doesn't understand the previous explanation. These systems, which include [Hov88,MP93,WAB91], use planning operators that reason explicitly about a system's intentions in presenting content and how those intentions can be achieved.

As it commonly happens, there is a trade-off between the two approaches. Schemata are less powerful, but are easier to write than plan operators, and planners using schematas are generally more efficient than plan-based text planners [LP95]. On the other hand, the latter are more principled, but they are still at the prototype stage. Moreover, as [YM94] points out, plan-based text generators have rarely if ever been formally assessed in terms of soundness and completeness, and the basis for writing plan operators has often been researchers' intuitions rather than more solid motivations. Finally, note that, while the two approaches are definitely different, they are related in that schemata can be seen as precompiled text (sub)plans.

A common characteristic of the two approaches just described is that the structure of the text is planned top-down. There are also local approaches such as Sibun's [Sib92] that can be seen as bottom-up; in fact, [Sib92] takes an even more radical approach to text planning, as she does away with building the global structure for the text altogether. Sibun argues that the hierarchical structure present in some texts, in particular descriptions of highly structured domains such as house layouts, family trees, etc, just reflects the domain itself, and not necessarily requires to be planned by a text planner; rather, such a text can be generated with local strategies. While Sibun's local strategies seem appropriate for the kinds of texts her system generates, it is not clear how they could be applied to texts that don't reflect the domain structure so closely; on the other hand, her approach is related to the need of encoding discourse communication knowledge (DCK for short). [KKR91] argues that DCK is a third level of knowledge that a NL generator should encode, intermediate between domain knowledge and communication knowledge. We will come back to the different approaches to text planning and to discourse communication knowledge in Section 7.1, when we will discuss these topics in relation to generating technical orders.

6.1.2 Sentence planning

Text planning selects and organizes the propositions, events and states that should be described in a discourse, but a second intelligent process is needed to generate semantic and syntactic specifications that can actually be realized in natural language. This process is called sentence planning [KKR91]. The key function of sentence planning is to select and adapt linguistic forms so that they are suitable for the local context in which they are to appear.

One important issues in sentence planning is to determine the content of definite noun phrases that are able to refer uniquely to an object and thereby allow a hearer to identify the object quickly and naturally. Research that addresses this problem includes [App85,DH91,Rei91,RD92]. Names for machine-internal representations of objects must be replaced with key properties of the object. The content needed to refer uniquely to an object varies greatly, according to the salience or attentional availability of the object in context, as well as the salience of distractor objects with similar properties. In the best case, the object is the most salient entity of the

appropriate type, and it can be described using a pronoun---with almost no content at all. On the other hand, if the object has not been mentioned in discourse at all, it may be necessary to provide a "complete description" of the object, including detailed information about its size, color, location and type. In the range of cases in between these extremes, the system must construct a reduced description, which should contain enough information to uniquely identify the object but should also be brief, so that the discourse remains clear and concise.

Of course, material meant to satisfy other purposes than unique referability may be included in a noun phrase as well -- for example, to convey an object's location to the hearer so they know where to look for it [App85], to convey an object's condition to the hearer so that they either take appropriate precautions (e.g., "hot") or do what's appropriate to achieve that condition (e.g., "well-oiled"), etc. The assumption is that this material will have been selected during text planning: sentence planning decisions involve whether to allocate the information to separate sentences (producing more, but perhaps simpler sentences) or to include in noun phrases (producing fewer sentences, but perhaps involving more reasoning on the part of the hearer).

Similar choices appear to be involved in building other sentential constituents [SD96]. Like descriptions of entities in noun phrases, descriptions of actions in verb phrases, descriptions of events in dependent clauses, and temporal and spatial locations in adverbials vary depending on the context. A clear and concise discourse will include enough information about them so that the hearer can determine which is meant, but not so much that the text is confusing or hard to read. Moreover, generating descriptions serves as a natural paradigm for integrating a variety of choices about which lexical items and which syntactic structures should be used to realize a sentence. It is a useful software engineering move.

Sentence planning must be a separate stage from text planning for two reasons. The first is its dependence on salience, which is determined both by the structure of discourse and the syntactic and semantic structure of immediately preceding sentences. Salience is not available during text planning. The second is the basis for including content, which differs from that of text planning. In text planning, content is included based on complex, abstract operators that organize propositions into arguments. In sentence planning, as noted above, content is included in referring expressions by simple operators that add small units of information, so as to identify objects or achieve in a terse manner goals identified during text planning.

Nevertheless, like text planning, sentence planning requires a rich representation of the world and the hearer. World knowledge must provide an inventory of properties about each object that can be included in descriptions. Hearer knowledge must include how the hearer can use such properties to rule out alternatives to the object being described.

6.1.3 Surface realization

The final stage of generation is surface realization. This is a purely linguistic level, which takes choices about words and syntactic structures made during sentence planning, and constructs a sentence using them. This relatively well-understood process is effectively analysis in reverse, and involves applying morphological and phonological rules as well as reversing the syntactic grammar [SvNPM90,YMVS91,PS93].

6.1.4 Specifying needed information

A variety of reasoning formalisms are available which can represent the knowledge needed in text planning and sentence planning and can capture the needed inferences. The hard part is formalizing the actual knowledge needed for particular problems. A graphical language for specifying processes is a natural medium for streamlining and automating as much of this formalization as possible. There are two reasons why this is a promising area of research.

First, we can associate the constructs of a graphical language with logical descriptions automatically. This can be done across the board for simple graphical languages with precise semantics [BALC95]. It can also be done by composing programs from larger chunks whose logical descriptions are well-defined.

Second, we can easily augment the graphical language with interactive tools for specifying and editing the logical descriptions themselves. As in work like [MFCS87,EPM93], a user-interface that constrains its input to match an ontology of possible specifications can simplify the task of obtaining high-level knowledge about a process and improve the accuracy and efficiency of building knowledge-intensive systems.

6.2 Action Descriptions and Instructions

In considering text, the first distinction that must be drawn is between actions, which are things that happen in the world, and action descriptions, which are text strings -- descriptions of actions in a language. The same action can be described by many different text strings, and many text strings can contribute to the description of one action. The description one chooses to give of an action depends on many things, including how much one knows of the action, the purpose of describing the action to a given audience, and the knowledge and skills of that audience.

In considering action descriptions then, a second distinction that must be drawn is between narrative and instructions. Roughly, narrative text describes what has happened (enriched by scene-setting descriptions, descriptions of characters and their motivation for acting, etc.), while instructions describe what should be done to achieve one or more specific aims in a given situation. How much is described explicitly and how much is left implicit, depends in part on beliefs about the audience's knowledge and skills. In the latter case, one relies on the audience's knowledge and skills to fill in what is necessary to behave appropriately [SC88,WD90,WBB92]. For example, consider what the audience for the following (separate) instructions must know to carry out them out correctly:

1. Carry the beakers carefully.
2. Go to the mirror and fix your hair.
3. Get me a cold soda.

In the first example, a hearer must derive from "carefully", behavioral constraints that will keep the beakers from breaking and their contents from spilling. In the second example, the hearer must derive from the phrases "to the mirror" and "fix your hair", that the appropriate location in front of the mirror is one that will enable him to see his hair clearly. In the third example, if a cold soda isn't immediately to hand, the hearer must derive from "a cold soda" locations where one might be located.

It is not that equivalent inferences can not be drawn from corresponding declarative sentences, such as one might find in a narrative:

1. Mary carried the beakers carefully.
2. Fred went to the mirror and fixed his hair.
3. Fred got Mary a cold soda.

it is just that to behave appropriately, such inferences must be drawn, if the "missing" information is not otherwise provided.

Many different kinds of information are provided in instructions, sometimes integrated into a single text, sometimes separated out and placed in identifiable locations or structures within a text. These types of information include:

- Procedural instructions - Basic action descriptions such as

Open 3L door and install support rod into upper attachment.

- Warnings - General statements about undesirable and/or dangerous consequences of performing procedural instructions in certain ways or under certain conditions. Warnings may be absolute (as in the first sentence below) or conditional (as in the second):

  Do not operate equipment for more than 30 minutes without cooling air to prevent damage to electronic equipment. If power is to be reapplied without cooling air, allow a 15-minute cool-down period.

- Context Specifications - Descriptions of the state of the world, including available resources, that either hold (thereby motivating compensatory actions or manner, as in the first example below) or should hold (thereby motivating constructive actions, as in the second example below) in order for the instructions to make sense and have their intended results

  Note: The unit weight is approximately 185 lb. and requires a four man lift.

  With door opened, adjust switch until roller contacts cam and continuity is indicated at pins A and B.

- Explanations - Specifications of why things should be done in a particular way, including in terms of the possible consequences of doing it in other ways.

  Do not use a blowtorch -- it is very easy to start a fire, especially in a loft.

As the final example shows, more than one element (here, warning plus explanation) may appear in a single sentence. While the focus of this report is on procedural instructions, we will note requirements for explanations and warnings, as necessary (albeit not sufficient) for ensuring that actions are performed appropriately.

In considering the generation of procedural instructions, there is one final distinction that must be made before noting the elements of a procedural instruction: that is whether instructions are given prior to action or in the context of action. In the latter case, Cohen [Coh84] has observed significant interleaving of fine-grained requests or commands on the one hand, and acknowledgments of understanding or requests for clarification on the other. For example, he notes

> In the present study, there were at least two requests used for each assembly step in Telephone mode. Each pair of requests (identification requests followed by assembly requests, or requests to pick up followed by assembly requests) involved at least one common object being manipulated. ([Coh84], p. 110).

Cohen observed that identification requests, where the speaker requests the hearer identify the referent of a noun phrase, dominate the first mention of an object in spoken instruction-giving discourse. These rarely had the form of an explicit request such as "Find the yellow tube." but rather took the form of an existential proposition -- e.g. "There's a little blue cap."; a perceptual request or statement -- e.g. "You'll see three very small pieces of plastic."; or a sentence fragment -- e.g. "Now, the smallest of the red pieces". Subsequent reference to an object was, as in written text, through pronouns and definite noun phrases.

Instructions given prior to action rarely have this form. More often, they are given in the form of steps consisting of one or more utterances. For example,

> With door opened, adjust switch until roller contacts cam and continuity is indicated at pins A and B. Verify positive switch contact by tightening bottom nut one additional turn.

While there are no firm guidelines as to what a single instruction step should encompass, often steps are organized around small coherent sub-tasks (such as adjusting the switch, in the example above). A step may specify several actions that need to be performed together (possibly in some partially-specified order) to accomplish a single subtask, or several aspects of a single complex action (e.g. its purpose, manner, things to watch out for, appropriate termination conditions, etc.). The important point is that an agent must develop some degree of understanding of the whole step before starting to act, so that he knows what all to do and what all to attend to.

The issues then that one needs to consider in generating procedural instructions include the following:

- Lexical Choice - in particular, the choice of verb used in describing an action (see Section 7.2);
- Action Purpose - expressed either in terms of intentional relations between actions or causal relations between actions and conditions. Specifications of purpose are extremely important in procedural instructions: they are used both to explain and justify an action and to implicitly convey features of how an action should be performed. The latter relies on the agent's using its knowledge and skills to recognize how the specified action should be carried out so that it satisfies the specified purpose.
- Termination Conditions - since actions are often best described in terms of the process an agent should carry out (e.g., "rotate", "push", "spray", "pressurize", "heat", etc.) and processes have no intrinsic end point, procedural descriptions in terms of processes must have a termination condition explicitly or implicitly specified. There are many ways of specifying a termination condition explicitly, including the use of "until" clauses, purpose specifications, and "enough to" specifications.

6.3 Hypothesis

We hypothesize, in the case of tasks represented as structured collections of PaT-Nets, that PaT-Nets at every level above the simulation level can effect choices in Text Generation. This hypothesis needs to be refined in terms of which sub-tasks within text generation may be affected.

7. Text generation for Technical Orders

In this section, we examine the problem of text generation that we discussed in general in Section. 6 in the context of the specific task of generating technical orders. Recall that, as discussed earlier, the process of text generation is divided into text planning, sentence planning, and surface realization.

7.1 Text planning for Technical Orders

In Section 6.1.1, we discussed three different approaches to text planning: those based on schemata [McK85,LP95]; those based on planning by means of plan operators [Hov88,MP93,WAB91]; and those that don't plan for the global structure of the text, but generate text incrementally by means of local strategies [Sib92].

We already mentioned in Section 6.1.1 that a local approach such as Sibun's doesn't seem appropriate for technical orders: her approach seems to be most effective when the structure of the text is not deeply nested, namely, when there are many equivalent objects to be described at the same level of embedding. However, technical orders are both more deeply embedded, and more rigid in structure, as we will show in Section 7.1.1: in particular, the choices that arise concern the macro structure of the text, rather than specific objects to be included.

It is our opinion that a schema based text planner is more appropriate for NL generation from PaT-Nets than a full fledged plan based text planner, for the following reasons.

- First, the text to be produced is not free instructional text, but it is in the form of technical orders. Technical orders have a somewhat predefined, rigid structure, discussed in Section 7.1.1, that lends itself to be encoded as discourse schemata. In fact, some of the knowledge about the high level structure of technical orders is akin to discourse communication knowledge a lá [KKR91] --- see Section 7.1.2.
- Second, NL generation from PaT-Nets is not performed within an interactive application, but rather, within a (possibly semi-) automatic support system for technical order writers; thus, issues for which the intentional structure of discourse is crucial, such as replanning to answer follow up questions, are not as relevant as in e.g. an interface to an expert system that has to explain how it reached certain conclusions.
- Third, building a new plan-based text planner requires more effort than building a schema-based one; planners such as the one in [MP93] are available, but its authors themselves criticize it as not being principled enough.
- Fourth, PaT-Nets provide a well specified and constrained domain representation, that can be seen as providing a skeleton of a text plan in itself. For modularity, it is not advisable to generate directly from PaT-Nets, but rather, it is better to interpose a level such as that of discourse schemata or plan operators between the text planner and the domain representation; however, schemata may be sufficient for this application.
- Finally, NL generation from PaT-Nets seems to require careful consideration of the two following issues, rather than of text planning in itself:
    - the level of detail at which instructions have to be generated, which includes decisions e.g. about whether to describe specific arcs and subnetworks;
    - sentence planning, both at the level of sentence structure, and at the level of lexical choice. In fact, while the global structure of technical orders is somewhat rigid, as we will show in Section 7.1.1, the structure of the sentence seems to afford more flexibility. For example, in the F16 instructions, we find the well-known alternation of purpose and means clauses. An example of purpose clause is the infinitival clause in italic:

To disconnect hydraulic test stand from system A, perform steps 12 through 15. For system B, perform steps 16 through 18.

An example of means clause is the following by + -ing construction (in italic):

All personnel engaged in refueling shall discharge electricity from their persons by touching a static ground cable or grounded object before each operation.

These two constructions can often be used interchangeably in instructions; the previous example could be rephrased as follows, by transforming the main clause in a purpose clause (in italic), and the means clause in the new main clause:

To discharge electricity from their persons, all personnel engaged in refueling shall touch a static ground cable or grounded object before each operation.

However, while the two clauses are semantically interchangeable, one form is often preferred to the other because of other constraints on their usage that depend on the surface organization of text, as discussed in [VM95]. Studying these kinds of constraints affecting surface structure appears to be more pertinent to generating technical orders than a detailed study of text planning strategies.

In conclusion, a schema based text planner (where schemas can be regarded as precompiled directives for text structure) may be appropriate and less time consuming to build than either building a plan-based DP from scratch, or adapting one of the existing ones.

7.1.1 High level structure of technical orders

The F16 technical orders we have examined are subdivided into two main sections: the introduction, that describes some general procedures such as connection of electrical power; and a section on maintenance procedures. We will discuss the organization of the latter, as the procedures described in the former follow the same format in a simplified way.

The procedures ρ described in the maintenance procedure section of the F16 technical orders all (?) concern the substitution of a certain Component X, and are organized as subdivided into the three subprocedures removal, installation and checkout of the Component X in question. That the three subprocedures removal, installation and checkout are not independent, but are part of the global task of substituting Component X --- never explicitly mentioned! --- is shown by the fact that the first part of each ρ, Input conditions, consists of conditions, recommendations and safety notes that apply to all three subprocedures. Note that removal, installation and checkout are labeled with (1), (2) and (3) respectively in the technical orders, so that they can be referred to by means of these indexes.

We will discuss here the various subcomponents of a procedure ρ, which are, in order, input conditions, the three subprocedures, and follow-on maintenance.

7.1.1.1 Input conditions

It is the first part of ρ, which, as the name says, provides information on the state the world must be in for ρ to be executable and on the necessary resources. Input conditions consist of the following items, that we present in the order in which they appear in the technical orders.

1. Applicability. It details the types of aircraft to which ρ applies (as described under 4. on the second page of the F16 technical orders). The value is All if ρ applies to all types of F16.

2. Required Conditions. It is a list of all conditions that have to hold (possibly by being brought about) before ρ or one of its subprocedures --- (1) removal, installation, (3) checkout --- can start; if the condition doesn't apply to the whole ρ, the condition is indexed by the number(s) corresponding to the subprocedure(s) it applies to. For example, the following are the Required Conditions for Crossfeed valve, 2823FV4, Removal, Installation, and Checkout:

   - Aircraft safe for maintenance (JG10-30-01)
   - (1, 2) Aircraft defueled (JG12-10-02)
   - (1, 2) A1 tank drained (JG12-10-02)
   - (3) Aircraft fueled to approximately 2500 pounds (JG12-10-01)
   - (3) Access door 3308 open (General Maintenance)

   The first condition applies to the whole ρ, the second and third to Removal and Installation, and the fourth and fifth to Checkout. All these conditions refer to other parts of the job guide (?), such as (JG12-10-02).

3. Personnel recommended. First, the number of technicians required is stated (e.g. 7); then the activity of each technician is described, possibly preceded by the index of the subprocedure(s) the technician is involved in. For example,

   - (3) Technician C assists in checkout (forward cockpit).

4. Support Equipment. It is a list of equipment items, possibly preceded by the index identifying the subprocedure(s) the equipment is needed for.

5. Supplies (Consumables). It is often empty.

6. Safety Conditions. WARNING and CAUTION, in this order, are listed under this heading. WARNING concerns possible injury to personnel; it may also include information about damage to the aircraft. CAUTION concerns only possible damage to the aircraft.

7. Other recommendations is a list of pairs, Item - Purpose, where Item is the code identifying another procedure, and Purpose the reason why Item is performed. For example

   Other Recommendations:
   Item
   Purpose
   1-1-3
   To purge fuel tank

7.1.1.2 Subprocedures

The three subprocedures (removal - installation - checkout) are organized as sequences of sequentially numbered steps. Each step may be indexed by the alphabetical identifier of the technician that has to perform it. E.g. Step 1 of subprocedure Removal of Fuel Flow Proportioner has to be performed by technician B:

1. (B) Depressurize hydraulic systems A and B. (General Maintenance)

Interspersed through the steps, and preceding the step(s) they refer to, are WARNINGs, CAUTIONs, and NOTEs. WARNINGs and CAUTIONs are used as described above, while NOTEs are used as follows:

- To identify under which conditions certain steps have to be executed, e.g.

  NOTE
  Omit steps 28 through 31 if original FFP [Fuel Flow Proportioner] is to be reinstalled.

- To specify constraints.

  - Constraints may be global to the whole subprocedure --- in this case the NOTE appears at the beginning of the subprocedure, before the sequence of steps starts:

    NOTE
    All serviceable parts will be retained for reinstallation.

    - Constraints may apply only to a subsequence of steps:

      NOTE
      Petrolatum shall be used for lubricant when required in following steps.

    The subsequence may be composed of a single step: the following NOTE provides a constraint on the execution of the action install, described in step 3 (recall that (A) in step 3 refers to technician (A)):

      NOTE
      Screw shall be installed in lower outboard corner of pump and support.

      3. (A) Position support on pump and install three bolts and screw. Torque three bolts and screw to 40-60 inch-pounds.

- Finally, a NOTE may describe actions that shouldn't or couldn't be performed in connection with the step following the NOTE, 7 in this case:

  NOTE
  Upper outboard bolt cannot be removed at this time due to interference by lower engine supply fuel tube and supply elbow.

  7. (A) Remove three bolts (two inboard and lower outboard), three washers, and three sealing washers. Discard three sealing washers.

The last component appearing in subprocedures is RESULT. RESULT refers to the immediately preceding step S(i), and describes a condition that S(i) either brings about (as for step 19 below) or is used to verify (as for step 20 below) --- step 19 and 20 are part of the same subprocedure (checkout of fuel/oil heat exchanger):

> 19. (B, C) Increase hydraulic system A pressure to 3000 psig as indicated on HYD PRESS A indicator.
> RESULT:
> Ground test panel FFP advisory light comes on. (28-23-DF)
>
> 20. (A) Inspect all disturbed connections for leakage.
> RESULT:
> No leakage allowed. (28-23-FD)

If a RESULT concerns more than one condition, there will be a list of such conditions under a single heading RESULT. Presumably the reference codes in parentheses refer to what should be done in case the expected condition doesn't hold.

Note that the structure of the procedures described in the first section of the manual is the same as that just described: namely, they include WARNINGs, CAUTIONs, NOTEs and RESULTs, but they don't include Input conditions and Follow-up maintenance.

7.1.1.3 Follow on maintenance

This is the last part of a procedure ρ. It is a list of actions or procedures to execute, possibly with the indication of the subprocedure they refer to, e.g.

(3) Refuel aircraft (JG12-10-06)

7.1.1.4 Stylistic features

Different components of ρ are expressed in different ways. More specifically, steps are expressed by means of imperative, including negative imperatives such as Do not torque. Each step is composed of up to five action descriptions; connectives such as and then are used. Purpose and Until clauses occur.

The style is somewhat telegraphic. This in particular affects Noun Phrases (NPs), that never include articles: therefore NPs may be ambiguous between definite anaphora and indefinite reference, as in (from Installation of Fuel Flow Proportioner):

1. (A) Lubricate two packings (MS28778-4) with hydraulic fluid and install on two unions.
2. (A) Install two unions. Torque to 143-158 inch-pounds.

The issue is whether two unions in 2. refers to the same two unions in 1 --- this seems to be the case, given the concise style used in technical orders. However, note that normally coreference should be expressed by means of a definite article, as in the two unions.

In WARNINGs and CAUTIONs declaratives are used; modals such as shall and must frequently occur.

WARNING.

Personnel working near aerial refuel receptacle shall use care to prevent personal injury from inadvertent operation of slipway door.

Occasionally, in CAUTIONs imperative are used:

Do not operate equipment for more than 30 minutes without cooling air to prevent damage to electronic equipment. If power is to be reapplied without cooling air, allow a 15-minute cool-down period.

NOTES are the most varied in style: some are expressed as imperatives, others as declaratives.

7.1.2 Domain communication knowledge

In [KKR91], Kittredge et al. discuss the notion of domain communication knowledge (DCK for short). They argue that DCK is a third level of knowledge that a NL generator should encode, intermediate between domain knowledge and communication knowledge: DCK is not needed to reason about the domain, but rather, to

communicate about the domain. To characterize the difference between DCK on the one hand, and communication knowledge and domain knowledge on the other, they provide the following arguments.

Communication knowledge is the knowledge embodied in the schemata or plan operators that we discussed earlier [KKR91] illustrate the difference between DCK and communication knowledge by the problem of planning object descriptions:

Consider the task of describing a set of objects in some domain. Communication knowledge about thematic structure implies a strategy that describes together those objects that share some feature. Domain knowledge can supply information about which objects share which features. But if there are many different features, the task remains of choosing the feature(s) according to which the descriptions will be grouped together. This choice must be based on knowledge that is neither general knowledge about communication (since the choice depends on the particular features of objects in the domain) nor actual domain knowledge (since it is only needed for planning communication).

As the difference between DCK and domain knowledge is concerned, [KKR91] discuss the difference between a police crime report and a detective novel.

> The underlying domain knowledge is knowledge about events related to the crime, along with general knowledge about human beliefs, desires, intentions and actions. Nonetheless, the two texts generated from this knowledge are quite different. Clearly, the domain knowledge itself does not imply a particular way of communicating about itself.

In the rest of their paper, Kittredge et al. discuss how DCK comes into play in specific kinds of reports, one regarding marine weather forecasts, the other regarding employment statistics. For example, in both cases the order in which the information is presented is fixed. In the weather reports, warnings about bad weather precede the main body of the report. The reports about employment statistics are composed by an overview, followed by multi-paragraph blocks about employment and unemployment respectively. In turn, these two blocks are subdivided into a summary paragraph, followed by paragraphs devoted to changes in (un)employment organized around various parameters (e.g. by sex and by age).

One drawback of the [KKR91] paper is that there is no discussion of how DCK could be expressed in practice. Some of Sibun's local strategies [Sib92] are an example of an explicit encoding of DCK.

7.1.2.1 DCK in Technical Orders

One instance of Domain Communication Knowledge in Technical Orders is the structure of Required Conditions and Follow-on Maintenance. Required Conditions occurs at the very beginning of ρ and includes all the conditions that have to hold before ρ or one of its subprocedures executes; Follow-on Maintenance occurs at the very end of ρ and includes all the follow-on maintenance actions, even if they concern only one of ρ's subprocedures.

Clearly, the way these conditions are expressed does not belong to general communication knowledge, that concerns the macro structure of text. Moreover, it does not belong to domain knowledge either. Consider Required Conditions. In our process representation, conditions that affect different transitions, or different nets, are associated with the corresponding transition or net. Note in fact that it is wrong to associate all the conditions belonging to Required Conditions with the global ρ, as some of these conditions are mutually exclusive, namely, they can't hold at the same time. In the example on page , one of the conditions that applies to Removal and Installation (Aircraft defueled) clearly can't be true simultaneously with one of the conditions relative to Checkout (Aircraft fueled to approximately 2500 pounds). Thus, the discourse planner will have to collect the appropriate conditions from the process representation and group them in the same portion of the text; this is analogous to, in [KKR91] example of marine weather reports, rendering the dangerous weather conditions as warnings that all appear at the beginning of the report.

7.2 Lexical Choice in Generating Maintenance Activity Descriptions

We are currently engaged in a lexico-syntactic analysis of the verbs that occur in task order descriptions. This information will include the subcategorization frame or frames for each verb sense, with corpus-based frequency counts, the selectional restrictions on the associated verb arguments, and verb class membership in a specially constructed domain specific verb lattice. Having this amount of detailed information about the verbs will enable us to choose the appropriate lexical item when generating an English description from an instance of a parameterized action.

```
1. Let Act be a parameterized action instance
2. Let LexicalEntries be a list of (syntactic frame, parameterized action) pairs
3. Let LE be the pair in LexicalEntries whose parameterized action part best
   matches Act in terms of features (ignoring parameterization differences)
4. Let A be the parameterized action of LE
5. While there are still unfilled features in A that are filled in Act
     Let D be an empty parameterized action instance
     For each unfilled feature, Fname, in A that is filled (with Fvalue) in Act
         Fill the Fname feature of D with the value Fvalue
     Let NLE be the pair in LexicalEntries whose parameterized action part
       best matches D
     Combine LE with NLE, unifying A with the parameterized action of NLE
       (if unification fails, then do not change LE) and combining the
       syntactic frames
```

Figure 13: Representation to NL instructions control algorithm

Figure 13 shows the algorithm to translate parameterized actions into natural language instructions. The lexical item chosen in Step 3 will be the one that most closely matches the action, in several different areas, including:

- the essence of the action, i.e.,
    - the movement or manipulation that is involved, i.e., grasping vs lifting,
    - the attribute/value modification involved, i.e., disengage vs open
    - cognition involved, i.e., note
    - sensing involved, i.e., inspect vs touch
- applicability conditions, i.e., in order to remove a gas cap, it must first be present
- post-conditions, i.e., the successful completion of a push action involves a change of location of the object.
- the number of arguments involved, i.e., presumably an agent as well as possibly one or more objects
- type constraints on those arguments, i.e., refueling necessarily involves an appropriate type of fuel for the device in question

The difficulty in choosing the lexical item will vary significantly from situation to situation. The most difficult choice occurs when we are given a sequence of PaT-Nets, with no clear grouping into individual actions. Trying to determine which PaT-Nets should be treated as coherent units is a combinatorily explosive search problem. However, if we can assume that the interface that created the PaT-Net sequences in the first place bracketed separate sequences appropriately as they were being generated, then our task becomes more feasible. There are still many different scenarios, depending on the type of action involved, and how specific its implementation is to the objects it is being applied to.

7.2.1 Object oriented actions

The choice of a lexical item for a verb can be completely determined by an object-specific script for a noun involved in the action, as in the table in the appendix. For example, the description of an open action will depend on the particular object that is being opened (a valve vs. an access panel); in this case, the object contains a named script for open, which determines the lexical item to choose. However, when such a script is not available, as is often true for low-level actions, or when the action is more widely applicable, a generic description of the action must be used.

7.2.2 Generic action descriptions

In recent linguistic literature, efforts have been made to classify verbs in terms of their semantic properties (Levin 1993, St. Dizier 1995, among others). The goals of that research have been to identify semantic factors which influence and correlate with syntactic behavior. This has resulted in the identification of useful components of meaning, which are on the one hand linguistically encoded in structures such as the lexical entries of verbs or grammatical constructions, but have on the other hand great relevance for the representation of actions in the world. These components include:

- exertion of force: requires a magnitude of force which in turn can affect speed and distance of change of location
- directed motion: requires a trajectory
- contact: requires respective location points
- change of location of an object: requires a path for the object

In attempting to generate an unambiguous, detailed, English description from an animated simulation, the identification of such meaning components, as in the example below, provides crucial constraints that can significantly reduce the complexity of the task. Verbs can be represented in a lattice that allows semantically similar verbs, such as all motion verbs, to be closely associated with each other, and with a higher level node that captures the properties these verbs have in common. Entire clusters of verbs will correspond to the primitive actions such as MOVE and LOCOMOTE, GRASP and REACH. These generalizations combined with the unique properties of the verbs that allow them to be distinguished from each other are the very properties that are relevant to lexical choice.

A coarse-grained grouping of F-16 verbs can be made by dividing them into verbs that involve a change of state and verbs that do not. Verbs that do not necessarily change the state of an object include those requiring the Agent to check the status of the object using cognitive and sensory capabilities (verify, inspect).

Change-of-state verbs can be subdivided into simple changes of state involving one object, such as rotate or tighten or into more complex changes of state involving symmetrical changes to more than one object, such as attach or replace. Part of the categorization of these verbs can simply include the number of arguments involved, immediately ruling out several potential choices for a particular action.

For example, the task order description for attach always involves two objects and an agent, which is the implicit you of an imperative ("Attach cap chain hook to receptacle"), and can be categorized as a CONTACT verb.

```
attach(L-agent, L-object1, to(L-object2)))
(agent = L-agent;
 object1 = L-object1,
 object2 = L-object2;
```

```
 spatiotemporal =
        (goal = (execution-type = establish;
                 relation = contact(object1, object2))))

contact(X,Y) <- locpt(X, X1),
                locpt(Y, Y1),
                at(X1, Y1).
```

The representation for connect, another CONTACT verb, is similar to that for attach except that an additional intermediary may be involved, requiring a transitive contact relationship [Pal90] ("Connect static bond cable between fuel truck and aircraft"). In generation, the inclusion of the intermediary in a CONTACT relationship would result in the choice of connect rather than attach as the lexical item.

```
connect(L-agent, L-object3, between(L-object1, L-object2))
(agent = L-agent;
 object1 = L-object1,
 object2 = L-object2,
 object3 = L-object3;
 spatiotemporal =
        (goal = (execution-type = establish;
                 relation = contact(object1, object3),
                            contact(object3, object2))))
```

A more fine-grained classification of verbs can be made by focusing on the specification of the post-condition. Does the change of state involve a putting action, where an entity is directed towards a particular location, either with a specific instrument or motion as in funnel or push, or involving a specific type of entity (a liquid or mass noun) as in spill?

7.2.3 Motion verbs

The verbs that change the status of only one object include as a subset the class of motion verbs, where the changed state of the object is its position or orientation. All motion verbs require a spatio-temporal component describing the position of the object over time, and may include additional components (e.g., culminating conditions and manner of motion) that are specific to the particular verb or subclasses of verbs. For example, the motion verbs used in the F16 technical orders corpus can be partitioned into two main subclasses:

1. Verbs of inherently directed motion (enter, escape, leave, return). The semantics of these verbs include a specification of the direction of motion (even in the absence of an overt directional complement), but not the manner of motion.

2. Verbs specifying manner of motion (rotate, turn, slide, move). The meaning of these verbs include a notion of manner or means of motion, but no direction of motion or culminating condition.

Although in general many of these verbs can refer to either an agent's volitional change of location (I walked home, I turned around (primitive LOCOMOTE) or an agent changing the location of an object, I turned the chair around (primitive MOVE), in this domain the usages almost always correspond to the latter.

The syntactic and semantic behavior of these two subclasses of motion verbs are documented in (Levin 1993), see Figure 14, and this allows us to make generalizations about each subclass -- the arguments required as well as how these arguments may or may not be realized syntactically. The semantic components of a verb or verb class aid in searching for the proper lexical item to express an action, and the syntactic frames associated with the verb or verb class provide guidelines as to how to describe the action linguistically. Thus, knowing which Levin class a verb belongs to helps in both lexical choice and sentence planning.

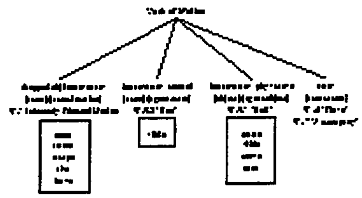


Figure 14: Portion of verb lattice involving Levin's motion verb classes

As a verb of inherently directed motion, enter has an underlying representation with two arguments: an object (object1) that is the actor undergoing motion, and an object (object2) that is the location that is entered. (When more than one object is involved in the motion event, the first object is understood to be the one undergoing motion, and other objects provide constraints on the motion of the first object.) The second argument may be expressed as an object of the preposition into or as a direct object (in the so-called preposition drop alternation), or it may not be expressed at all, if the context makes the location self-evident so that an explicit syntactic realization of the location would be redundant. The action is known to be completed once the proper geometric relation between the object and location is achieved (as in the case of enter) or terminated (as in the case of escape). Most verbs of inherently directed motion disallow an additional agent who causes the object to move ("sleeve will enter aerial refuel receptacle bore" is permissible, but not "enter sleeve into aerial refuel receptacle bore"). Consequently, these verbs are used in descriptions rather than instructions.

```
enter (actor, location)
- ----------------------
enter (L-actor, L-object)
enter (L-actor, into(L-object))
enter (L-actor)
(agent = <none>;
 object1 = L-actor,
 object2 = L-object
        OR
 object2 = <inferred>;
 spatiotemporal =
        (goal = (execution-type = establish;
                 relation = in(object1, object2))))
```

In contrast, verbs specifying manner of motion usually exhibit the causative - inchoative alternation in which the object undergoing the motion can be realized syntactically as either a direct object with an Agent causing the object to move ("slide bracket on shank") or as the subject of a sentence with no Agent specified ("bracket slides on shank"). In the absence of a prepositional phrase specifying direction, these verbs do not indicate any direction or goal of motion, although a particular verb may have additional arguments inherent in its meaning.

```
slide (agent, object1, object2) / slide (object1, object2)
- ---------------------------------------------------------
slide (L-object1)
slide (L-object1, on/over(L-object2))
slide (L-agent, L-object1)
slide (L-agent, L-object1, on/over(L-object2))
(agent = L-agent
        OR
 agent = <none>;
 object1 = L-object1;
 object2 = L-object2
        OR
 object2 = <inferred>;
```

```
spatiotemporal =
        (goal = (execution-type = maintain;
                 relation = contact(object1, object2)));
         kinematics = translational)
```

For example, the representation for slide includes an optional agent and an object1 that undergoes the sliding motion, plus a second object2 that provides the surface over which the sliding occurs. Although the direction of motion is not specified, any slide event must contain a translational motion component of object1 with respect to object2 and maintain contact between object1 and object2. The surface of sliding may be realized syntactically using an on-PP ("Slide sleeve on tube") or over-PP ("Slide coupling nut over packing and sealing ring"), or it may have to be inferred contextually ("Carefully slide matrix assembly forward to disengage two bushings"). Since "slide" is not an inherently telic verb (the sliding action may continue indefinitely), its end point should be specified explicitly ("sliding sleeve into housing"). The information from the goal clause ("into the housing") may also be used to supply the inferred sliding surface.

Another distinction that needs to made with respect to slide has to do with the amount of effort involved in moving the object, or the WEIGHT parameter. If the effort exceeds a certain threshold, then the action becomes a pushing action rather than a sliding action, and the choice of lexical item should change accordingly.

Some verbs used in task descriptions (reinstall, align, couple, coordinate, insert, replace, lubricate, refuel, discard, defuel, safety-wire, adjust, release, reposition) aren't in the Levin classes. One of our research issues will be determining appropriate classifications.

7.2.3.1 A detailed analysis of Grasp

To illustrate the potential benefits of our approach, we have provided below a detailed analysis of grasp, characterized in terms of the meaning components defined above. We describe the unique properties associated with grasp, and identify commonalties with other verbs which point to productive linguistic generalizations.

The transitive verb grasp

SUMMARY OF FEATURES

- Participants: Agent, (Instrument, default: hands), Patient
- Agent has intentionality
- Directed motion of Agent
- Agent comes into contact with Patient
- Conative alternation possible: negates implication of contact

We choose to consider the basic meaning of this verb to be an event of exerting force on an object, where the object is the denotation of the direct grammatical object of the verb. If the subject of the verb denotes an agent, such as a human being, the default assumption is that this agent exerts force on the object by closing his/her hand or hands around it. This definition corresponds more or less directly with the simulation primitive GRASP.

The basic action denoted by grasp under this definition is essentially punctual; the action of closing one's hand around an object is momentary and does not extend over an interval of time in the way that, for example, a carry event may. (The distinction between punctual events and other kinds of events, such as a carry events, is finally an issue of granularity, since no event is truly instantaneous. However, at the level of granularity required for this type of instruction understanding, the distinction is meaningful.)

However, it is clear that many types of events that may be described by the verb grasp involve more than just this simple hand-closing. Still, no event of grasping (except when grasp occurs in the conative alternation, as

discussed below) can fail to include this action. Therefore, we identify this as the core meaning of the verb when applied to an agent subject.

The meaning of an instruction such as (1), though, is not exhausted by this basic definition for grasp:

(1) Grasp handles.

Such an instruction may be intended to provoke a behavior on the part of the agent that includes reaching for the handles in question, and then also holding them in a certain way and for a certain length of time.

The extended meaning of grasp in cases like this may be productively viewed within the framework of an aspectual system such as that proposed by Moens and Steedman (1988). They note that an event in general consists of three parts: a preparatory process, a culmination, and a consequent state. A reference to an event, such as the reference in (1), may include some or all of these three parts, depending on contextual and other factors.

The preparatory process may involve bringing about the preconditions for the execution of the core action (culmination). In the case of (1), this means that the agent will reach for the handles, and thereby gets his/her hands in a position such that closing them will cause force to be exerted on the handles.

The consequent state refers to the desired post-conditions of the action, in this case, that the agent is holding the handles. Notice that the preparatory process is inherently bounded; it is over when the culmination of the event occurs. However, the consequent state is not bounded in this way. It is useful to speak of bounded processes as telic and unbounded processes as atelic. For example, (2) expresses a telic process:

(2) and carefully slide unit out from equipment bay.

since there will come a natural point at which this action has been completed, namely, when the unit is free from the equipment bay (as judged by either the agent or the instruction-giver). Verbs such as slide are not inherently telic because they could continue indefinitely. That is why the end point of the sliding event, being out of the equipment bay, must be specified.

In the absence of such an explicitly specified termination condition for a process, the agent carrying out the instruction in (1) will have to determine when the consequent state of the grasp is over. The default solution may be simply to wait until explicitly told to let go of the handles. Such an explicit instruction may take the form of (3).

(3) Stop grasping the handles.

In this case, the use of the aspectual verb stop, which applies only to processes, signals that grasping is referring to the consequent state of the grasp event. Another possible instruction to signal the end of the consequent state would be (4):

(4) Grasp the handles until I give the signal.

The until clause again refers to the consequent state of grasp, not the actual hand-closing action that is the core meaning.

However, it is more likely that an intelligent agent will interpret an instruction like (1) not as an end in itself, but as a step in a plan to do something with the handles. This reasoning about the larger context of a given instruction requires knowledge about the typical kinds of things that are done with handles, and involves expectations about subsequent instructions that may be given. An agent who reasons in this way will know something about how long to maintain the hold on the handles, particularly if holding the handles is a precondition to performing another subsequent instruction such as (2). Moreover, such knowledge is

independently necessary to constrain the many variations of grasp that an agent can perform, depending on the specific object to be grasped and the larger goal to which the grasping may contribute (Levison 1994).

The verb grasp can also occur in a syntactic frame containing a with adjunct, as is seen in (5).

(5) Grasp the handles with a monkey wrench.

Though the with phrase is syntactically optional, it does not pattern like other adjuncts, whose semantic contribution to the core phrase is an elaboration or specification. It more closely resembles the direct object of eat, which, though omissible in many contexts, is filling an essential role in the verb's predicate-argument structure (Palmer 1988). To see this, consider

(6) Grasp the handles.

This variant of (5), in its default reading, implies that the agent is intended to exert force on the handles directly by closing his/her hands around them. However, it is clear that in (5) the agent is not instructed to touch the handles with his/her hands (particularly not if the handles are very hot). This means either that the with adjunct has negated one part of the meaning of the core phrase it has attached to, which is not typical of an optional modifier; or else that the with adjunct is specifying the filler of an essential role in the meaning of grasp, which in the default case in (6) was taken to be filled by the agent's hands. This latter possibility resembles the situation with the optional object of eat; the default interpretation, when this object is missing, is that the role of the thing eaten is filled with some unspecified kind of food, but this may be overridden by an explicit object.

Under this analysis, both (1) and (6) have a syntactically unrealized argument position, corresponding to the semantic role of intermediary (Palmer 1993). The intermediary may also be viewed as the object of an implicit "manipulation subscene" (Gawron 1988); the agent manipulates the intermediary (the hands, or the monkey wrench) and this results in the object of the verb being affected in the desired way.

It may seem counterintuitive to view an agent as manipulating his/her own hands, but in the simulation environment, this has some validity, since instructions must be sent to the system to simulate hand motions as well as other kinds of actions. Also, non-human agents may have detachable end-effectors, so that the distinction between a hand and an instrument becomes hazier.

The alternative would be to view grasp and grasp with as two different verbs with distinct but related meanings, much as the transitive and intransitive forms of eat are sometimes considered to be distinct entries in the lexicon, in some theories of grammar. However, note that one may say

(7) Grasp the handles with your hands.

and this is basically synonymous with (1). (Synonymy in the case of instructions may be loosely defined in terms of the behaviors they provoke in otherwise equal circumstances.) This suggests that there is an implicit argument in forms like (1) and (6) that can be made explicit if desired, for clarity, without altering the meaning. The synonymy of (1) and (7) is problematic for an account where (1) is interpreted as not involving an intermediary at all. Possibly then (7) would receive an interpretation akin to that of

(8) The car hit the wall with its bumper.

but it is not clear that this differs from the intermediary interpretation; (8) could be interpreted as an example of non-agentive manipulation of an intermediary. The fact that examples such as (8) are only acceptable when the with phrase has as its object an inalienably possessed part of the subject, could be due to the fact that non-agentive manipulators, being unable to actively gain control over objects that they manipulate, must already be in a relationship of control with respect to those objects, for example, by virtue of being inalienably attached to them.

It is clear however that grasp is an agentive verb, requiring an intentional agent as its subject. In fact, it is more agentive than a verb like hit, since someone can be said to accidentally hit something, but it's odd to say that someone accidentally grasped something. This seems to derive from the semantics of the verb; while hitting typically requires only coarse-grained body manipulations, grasping involves carefully coordinated finger motions, and thus is seen to take more conscious deliberation. On the other hand, a falling person may instinctively grasp a tree branch without much intentionality.

The subject of grasp however is not an agent in the instrument subject alternation (Levin 1993), as in (9).

(9) The monkey wrench grasped the handles.

In an appropriate context, (9) could describe a situation where someone is manipulating monkey wrenches in such a way as to cause them to exert force on the handles. Not all instruments occurring in with phrases can be promoted to subject position in this way, however; (11) is not a valid paraphrase of (10).

(10) I grasped the handles with the thick towel.

(11) * The thick towel grasped the handles.

However, when an instrument does occur in subject position, this may be interpreted as a situation in which the true subject, the agent, has been omitted syntactically, and thus the instrument has been promoted to subject as the highest-ranked thematic entity remaining in the sentence (Fillmore 1968). We hypothesize that such omissions of the subject can only occur in very constrained circumstances, namely, where the omitted argument is immediately recoverable from discourse context.

8. Examples

In order to discover and work through the issues involved in text generation from our action representation, we looked at a simple example of putting gasoline into a car and what form the instructions would take. The list below shows the action representation instances for the example, ignoring applicability conditions and subtasks (see Section 8.1 below for a possible subtasks). Figure 15 shows the example in a PaT-Net-like form. Figure 16 gives the instructions that should be generated from our representation.

```
1.  (agent = you;
    objects = ( car.engine );
    culmination conditions = ((agent = you;
                                squery = "off(car.engine)"));
    spatiotemporal =
           (goal = establish;
            relation = ((obj = car.ignition;
                         oquery = "at(off-position)")))).

2. (agent = you;
    objects = ( car.gasIntake.cover );
    culmination conditions = ((agent = you;
                                squery="open(car.gasIntake.cover")));
    spatiotemporal =
           (goal = terminate;
            relation = ((obj = car.gasIntake.cover;
                         oquery = "closed")))).

3. (agent = you;
    objects = ( car.gasIntake.cap );
    culmination conditions = ((agent = you;
                                squery="free(car.gasIntake.cap)"));
    spatiotemporal =
           (goal = terminate;
            relation = ((obj = car.gasIntake.cap;
                         oquery = "in(car.gasIntake)")))).

4. (agent = you;
    objects = ( pump.nozzle, pump.lever );
    culmination conditions =
           ((agent = you;
             squery="free-from(pump.nozzle, pump.lever)"));
    spatiotemporal =
           (goal = terminate;
            relation = ((obj = pump.nozzle;
                         oquery = "on(pump.lever)")))).

5. (agent = you;
    objects = ( pump.lever );
    culmination conditions =
           ((agent = you;
             squery="up-position(pump.lever)"));
    spatiotemporal =
           (goal = establish;
            relation = ((obj = pump.lever;
```

```
                    oquery = "up-position(pump.lever)")))).

6.  (agent = you;
    objects = ( pump.nozzle, car.gasIntake );
    culmination conditions =
          ((agent = you; squery="in(pump.nozzle, car.gasIntake)"));
    spatiotemporal =
          (goal = establish;
           relation = ((obj = pump.nozzle;
                    oquery = "in(car.gasIntake)")))).

7. (agent = you;
    objects = ( gasoline );
    culmination conditions =
          ((agent = you; squery="desired-amount(gasoline)"));
    subtasks = ...).

8. (agent = you;
    objects = ( pump.nozzle, car.gasIntake );
    culmination conditions =
          ((agent = you;
            squery="free-from(pump.nozzle, car.gasIntake)"));
    spatiotemporal =
          (goal = terminate;
           relation = ((obj = pump.nozzle;
                    oquery = "in(car.gasIntake)")))).

9. (agent = you;
    objects = ( pump.lever );
    culmination conditions =
          ((agent = you; squery="down-position(pump.lever)"));
    spatiotemporal =
          (goal = terminate;
           relation = ((obj = pump.lever;
                    oquery = "up-position(pump.lever)")))).

10.(agent = you;
    objects = ( pump.nozzle, pump.lever );
    culmination conditions =
          ((agent = you; squery="on(pump.nozzle, pump.lever)"));
    spatiotemporal =
          (goal = establish;
           relation = ((obj = pump.nozzle;
                    oquery = "on(pump.lever)")))).

11.(agent = you;
    objects = ( car.gasIntake.cap );
    culmination conditions =
          ((agent = you; squery="in-position(car.gasIntake.cap)"));
    spatiotemporal =
          (goal = terminate;
           relation = ((obj = car.gasIntake.cap;
                    oquery = "in(car.gasIntake)")))).
```

```
12.(agent = you;
    objects = ( car.gasIntake.cover );
    culmination conditions =
          ((agent = you; squery="closed(car.gasIntake.cover")));
    spatiotemporal =
          (goal = establish;
           relation = ((obj = car.gasIntake.cover;
                          oquery = "closed")))).
```

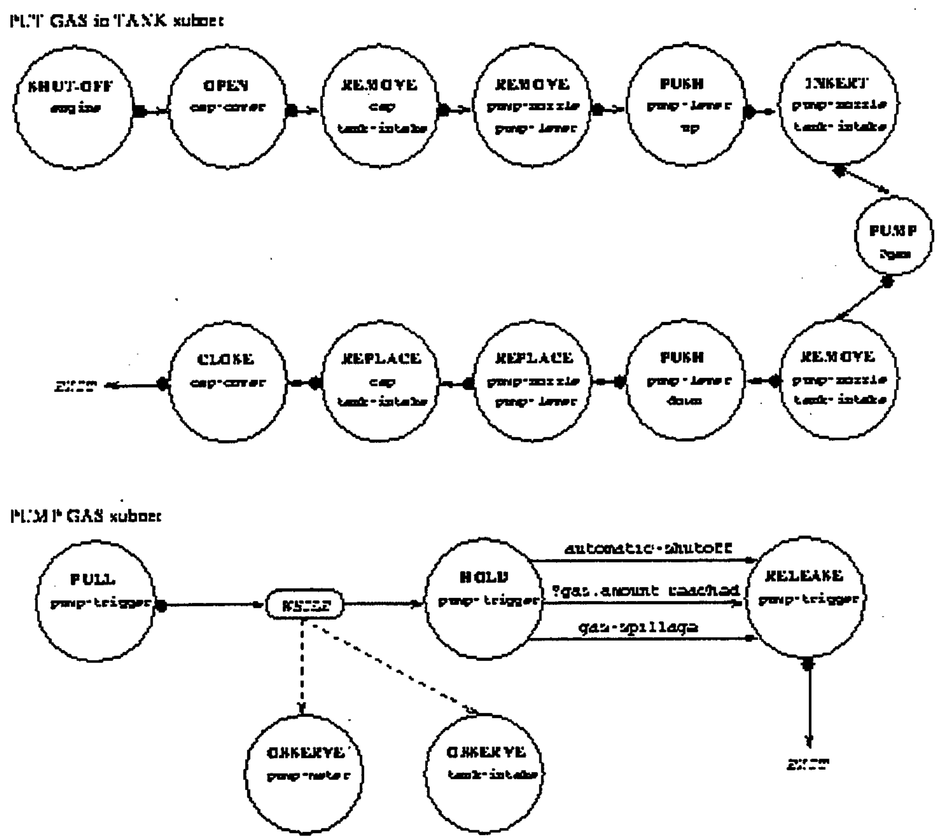


Figure 15: PaT-Net-like representation for the example

1. Shut off your car's engine.
2. Open the cover to your car's gas intake.
3. Remove the gas cap.
4. Remove the pump nozzle from the pump lever.
5. Push the lever up.
6. Insert the nozzle into the gas intake.
7. Pump the desired amount of gasoline.
8. Remove the nozzle from the gas intake.
9. Push the pump lever down.
10. Replace the pump nozzle.
11. Replace the gas cap.
12. Close the cover of the gas intake.

Figure 16: Natural language instructions for the example

8.1 Lexical Analysis for Fueling task

The natural language instructions for the fueling task include the following verbs: shut off, open, remove, push, insert, pump, replace, close, and the following objects: car's engine, cover, gas cap, car's gas intake, pump nozzle, pump lever, gasoline. In the analysis below, we have described each instruction in the task as a goal oriented predicate-argument structure. Our representation breaks the individual instructions down into conjunctions of low-level states that comprise the necessary components for the achievement of the goal. We introduced two new objects, the car door and the ignition so that we could describe the related actions of shutting off the engine and getting out of the car, which are precursors to the fueling action. Our aim in this analysis was to simply describe the necessary states, which would be achieved by individual PaT-Nets. The low-level states we introduced include: at, touch, state, grasp, pump, have, support. Each state corresponds to a simply binary predicate, except for have, which is in turn described as a conjunction of states, and corresponds to the notion of having physical control over an object.

```
have(object) <-
        at(hand,object),
        grasp(hand, object),
        state(object,LOOSE),
        support(hand,object).
```

The analysis begins here, and follows exactly the order of the natural language instruction. Notice that certain states (such as the car door being open) are added for coherence.

```
shut-off(engine,igntion) <-
        at(hand,ignition),
        touch(hand,ignition),
        state(ignition,OFF).

open(door) <-
        at(hand,lever),
        grasp(hand,lever),
        state(lever,OPEN),
```

```
        state(door,OPEN).

open(cover) <-
        at(hand,cover),
        grasp(hand,cover),
        state(door,OPEN).

remove(gas-cap) <-
        have(gas-cap).

remove(nozzle) <-
        have(hand,nozzle).

push(lever,up) <-
        at(hand,lever),
        touch(hand,lever),
        state(lever,up).

insert(nozzle,gas-intake) <-
        have(hand,nozzle),
        at(nozzle,gas-tank intake).

pump(gasoline) <-
        have(hand,nozzle),
        state(nozzle-lever,OPEN).
        pumped(gasoline).

push(lever,down) <-
        at(hand,lever),
        touch(hand,lever),
        state(lever,down).

replace(nozzle) <-
        have(hand,nozzle),
        at(nozzle,HOME).

replace(gas-cap) <-
        have(hand,gas-cap),
        at(gascap,HOME).

close(cover) <-
        at(hand,cover),
        touch(hand,cover),
        state(cover,CLOSED).
```

In the following chart we have associated specific, previously defined PaT-Nets with the states in the previous analysis. The table is object-oriented, in that the states are organized according to the objects that can occur in them.

Associated Object-Oriented PaT-Nets		
Object	State-Change	PaT-Net: object-specific
car door	CLOSED→OPEN	PUSH
	OPEN→CLOSED	PULL
gas-cap	HOME→LOOSE	LIFT-UP
	LOOSE→HOME	PUT-DOWN
	GRASPED→LOOSE	FIST
	LOOSE→GRASPED	UNFIST
	AT(LOC)→AT(LOC2)	WALKNETS
nozzle	HOME→LOOSE	LIFT-UP
	LOOSE→HOME	PUT-DOWN
	GRASPED→LOOSE	FIST
	LOOSE→GRASPED	UNFIST
	AT(LOC)→AT(LOC2)	WALKNET
nozzle-lever	CLOSED→OPEN	LEVER-PUSH
	OPEN→CLOSED	LEVER-PULL
	UNTOUCHED→TOUCHED	LAY FINGER UPON
cover	CLOSED→OPEN	COVER-PULL
	OPEN→CLOSED	COVER-PUSH
	UNTOUCHED→TOUCHED	LAY FINGER UPON
	GRASPED→LOOSE	FIST
	LOOSE→GRASPED	UNFIST
gas	UNPUMPED→PUMPED	PUMPNET

Chart 1: Associated Object Oriented PaT-Nets

8.1.1.1 Charts, Comparisons, and Data

Chart 2, below, represents verbs indexed by object and vice versa. In other words, we can see how much information can be gotten from either model:

Objects	Verbs								TOTAL
	shut	open	remove	push	insert	pump	replace	close	
engine	X								1
cover		X						X	2
gas cap			X				X		2
gas intake		X	X		X				3
pump nozzle			X		X		X		3
pump lever			X	X					2
gasoline						X			1
ignition	X								1
TOTAL	2	2	4	1	2	1	2	1	15

Chart 2: Indexed Verbs

Chart 4 shows where objects relate to primitive states and vice-versa:

Objects	at	touch	state	grasp	pumped	support	TOTAL
cover		X	X	X			3
gas-cap	X		X	X		X	4
gas intake							0
pump nozzle				X		X	2
pump lever	X	X	X				3
gasoline					X		1
ignition							0
TOTAL	2	2	3	3	1	2	13

Chart 3: Objects Relating to Primitive States

Chart 4 lists the primitive states that occur in the descriptions of each verb:

Verbs	Fundamentals						TOTAL
	at	touch	state	grasp	pumped	support	
shut off	x	x	x				3
open	x		x	x			3
remove	x		x	x		x	4
push	x	x	x				3
insert	x		x	x		x	4
pump	x		x	x	x	x	5
replace	x		x	x		x	4
close	x	x	x				3
TOTAL	8	3	8	5	1	4	29

Chart 4: Primitive States

On average, a verb breaks down into 3.625 fundamentals in this model.

On average, a fundamental appears in the description of 4.833 verbs.

9. Verbs

9.1 F16 verbs with frequency counts

The verbs in this list were collected from a corpus of approximately 100,000 words of technical orders for the maintenance of F-16 aircraft. The text was tokenized into words and the words were then annotated for grammatical category using a statistical part-of-speech tagger trained on instruction-genre materials from the Penn Treebank part-of-speech-tagged corpus. A morphological analysis on verbs was then performed, utilizing on-line dictionaries to identify the root forms of inflected verb tokens. The verb roots were then sorted by frequency of occurrence in the text. The counts for verb frequency do not include the occurrences of verbs in verb-particle constructions and adjectival predicates such as "turn off" and "be free of". Such constructions were identified semi-automatically by a collocational analysis of word n-grams with n set between 2 and 10 words. These instances of the verbs are counted separately. For example, of the 41 occurrences of the verb root "turn", 6 were cases where the verb was part of the verb-particle construction "turn off". Hence, in the counts below, "turn" is listed as occurring 35 times in isolation, and "turn off" is listed as occurring 6 times.

770	remove	646	install	536	note
452	position	340	require	276	torque
218	disconnect	210	use	197	connect
164	lubricate	160	perform	148	refuel
136	close	135	prevent	130	discard
128	omit	119	allow	108	open
87	follow	85	apply	80	verify
80	retain	75	move	75	engage
72	recommend	71	operate	70	secure
69	slide	68	purge	67	tighten
65	have	65	cool	64	match
61	seal	60	clamp	59	inspect
59	drain	57	rotate	53	check
51	result	50	outline	47	reinstall
47	indicate	47	give	44	prepare
41	be_open	41	act	40	assist
39	depress	38	defuel	35	turn
34	be_similar	33	compress	32	align
31	safety-wire	31	depressurize	30	stop
30	couple	30	adjust	29	provide
28	come_on	27	latch	27	increase
27	go_out	26	comply	23	slow
23	push	22	place	21	remain
21	lift	21	exceed	20	proceed
20	loosen	20	coordinate	19	insure
19	cause	18	start	18	insert
17	oscillate	17	occur	17	accomplish
16	wipe	16	list	16	fuel
16	attach	16	approve	14	replace
14	obtain	14	identify	14	contain
14	be_identical	13	release	13	mount
13	maintain	13	extend	13	avoid
12	lock	12	hold	12	clean
12	be_free_of	12	add	11	tag
11	service	11	press	11	fit

11 begin
10 facilitate
9 work
9 pass
9 exist
9 continue
8 locate
8 be_present
7 pack
6 wash
6 limit
6 illustrate
6 evaporate
6 come
5 support
5 permit
5 equip
4 unfold
4 spray
4 reduce
4 form
4 fail
4 control
4 be_serviceable
3 utilize
3 spill
3 seek
3 rest
3 monitor
3 key
3 enter
3 drop
3 designate
3 chock
3 be_visible
3 be_dangerous
3 back_off
2 unlock
2 terminate
2 speak
2 shorten
2 satisfy
2 pump
2 overcome
2 inhibit
2 face
2 delete
2 code
2 be_full
2 appear
1 vibrate
1 supply
1 stick
1 smear

10 reposition
10 collect
9 take
9 move_off
9 detect
9 conceal
8 disengage
7 touch
7 observe
6 turn_off
6 leave
6 hear
6 determine
6 bleed
5 seat
5 fold
5 disturb
4 trap
4 short
4 magnify
4 force
4 eject
4 cease
4 be_applicable
3 uncover
3 shut_off
3 saturate
3 preclude
3 mate
3 impregnate
3 eliminate
3 discharge
3 consist
3 blow
3 be_low
3 be_alert_for
3 attempt
2 transfer
2 switch
2 simulate
2 shim
2 refer
2 pull_out
2 measure
2 float
2 entrain
2 deform
2 bind
2 be_flush_with
2 amend
1 vent
1 supersede
1 spline
1 slit

10 read
10 affect
9 point
9 feel
9 decrease
8 separate
8 damage
7 repeat
7 change
6 retorque
6 insulate
6 grind
6 crisscross
6 be_clear
5 restrict
5 flow
5 associate
4 test
4 scratch
4 instruct
4 fill
4 define
4 break
3 wear
3 tee
3 shade
3 return
3 necessitate
3 make
3 evacuate
3 dry
3 direct
3 consider
3 bend
3 be_familiar_with
3 be_acceptable
3 actuate
2 trail
2 stabilize
2 show
2 select
2 raise
2 propel
2 lengthen
2 find
2 disseminate
2 cut
2 become
2 assemble
1 wait
1 uncouple
1 submit
1 splice
1 signify

1 shrink	1 sense	1 see
1 rise	1 retract	1 reset
1 report	1 reclean	1 receive
1 reaccomplishe	1 pull	1 pressurize
1 present	1 paint	1 notice
1 need	1 miss	1 minimize
1 mark	1 load	1 line
1 lead	1 keep	1 interfere
1 integrate	1 include	1 fit
1 exposee	1 escape	1 engage
1 end	1 empty	1 distribute
1 display	1 describe	1 depend
1 deactuate	1 cure	1 crush
1 cross-reference	1 create	1 cover
1 configure	1 concern	1 clear
1 center	1 cap	1 bypass
1 bond	1 bear	1 assure
1 assume	1 assign	1 achieve
1 accept		

9.2 Classification of Verbs with 10 Uses or More

Verbs that:

- change the status of multiple objects: remove, install, disconnect (?), connect (?), reinstall, align, couple, coordinate (?), insert, attach, replace
- change the status of one object: position, lubricate, refuel, close, discard, open, move, slide, purge, tighten, cool (1), seal, clamp, drain, rotate, depress, defuel, turn, compress, safety-wire, depressurize, stop (1), adjust, latch, increase, slow, push, place, lift, loosen, start (1), wipe, fuel, release, mount, extend, lock, clean, add, tag, press, reposition
- change and keep the status of objects: operate, cool (2), maintain, hold, service
- require readers to check the status of objects:, note (?), verify, inspect, check, insure, identify
- require readers to perform other tasks: prepare (2), obtain, collect, read
- specify tools and methods: require, use, prevent, allow (?), follow, apply, recommend, secure (?), provide, comply, avoid
- specify the status of readers' performance: perform, retain, act, stop (2), remain (1), proceed, start (2), accomplish, begin
- specify procedure: omit, result (1), outline, assist (1), list
- refer to the objects' status: be_open, be_similar, oscillate, contain, be_identical, be_free_of, fit
- refer to change of objects' status: come_on, go_out, remain (2), exceed, cause, occur
- refer to objects' mechanism: result (2), indicate, assist (2), facilitate, affect

Verbs which do not fit into any of these classifications: torque, engage, have, match, give, approve

9.2.1 *Analysis of some data from F16 corpus*

Representative examples of 25 "put" verbs from F-16 maintenance instruction corpus, with analysis of each example:

1. bind

" conical rubber seals shall be free of nicks , cuts , and tears and poppets shall move freely without **binding** or hanging . "

Form: "without" gerund
Subject: poppets (zero)

" if minimum lateral movement requirement is not maintained , doors may **bind** when operated . "

Form: "may"
Subject: doors

" depress relief valve poppet to verify it will not **bind** when operating . "

Form: "will", neg, S-comp
Subject: poppet (pronoun)

2. cap

" insure remaining elbow (_44f3 forward elbow or _44f4 aft elbow) is turned outboard and **capped** . "

Form: passive, S-comp
Passive Subject: elbow

3. cover, uncover

" remove housing **covering** fuel bypass valve . "

Form: gerund NP adjunct
NP Adjunct Subject: housing
Object: valve

" slide coupling nut back to **uncover** packing and sealing ring . "

Form: "to"-S
Subject: agent (zero)
Object: ring

4. face

" actuator indicator will **face** aft when properly installed on engine fuel shutoff valve . "

Form: "will"
Subject: indicator
AdvP: aft

5. fill

" allow sufficient time for cavity to **fill** prior to inspecting disturbed connections . "

Form: "for"-S
Subject: cavity

" if reservoir quantity is low , reservoir shall be **filled** as required to provide sufficient fluid for system pressurization . "

Form: "shall" passive
Passive Subject: reservoir

" if reservoir quantity is low , **fill** as required to provide sufficient fluid for system pressurization . "

Form: imperative
Object: reservoir (zero)

" cool oil will bypass fuel oil heat exchanger ; therefore , engine must dry-motor a minimum of 2 minutes to allow oil to heat and **fill** heat exchanger cavity . "

Form: "to"-S
Subject: oil (zero)
Object: cavity

" remove bolt from aft support and fit four washers on each side of aerial refuel receptacle mounting lug to **fill** gap between sides of aerial refuel receptacle mounting lug and aft support . "

Form: "to"-S
Subject: washers, agent?
Object: gap

6. fuel, defuel, refuel

" to minimize hazard of static electrical discharge , aircraft shall not be **fueled** when an electrical storm is within a 3-mile radius of servicing area . "

Form: "shall" passive, neg
Passive Subject: aircraft

" all support equipment not required for **refueling** shall be moved a minimum of 50 feet from aircraft . "

Form: "for" gerund

" (1,2,3,4) aircraft **defueled** (jg12-10-02) "

Form: passive NP adjunct
Passive Subject: aircraft

" (1,2) a1 tank **defueled** (jg12-10-02) "

Form: passive NP adjunct
Passive Subject: tank

" (3) aircraft **fueled** to approximately 2500 pounds (jg12-10-01) "

Form: passive NP adjunct
Passive Subject: aircraft
PP ("to"): 2500 pounds

" (3) aircraft **fueled** to between 200 and 350 pounds in forward and in aft tanks (jg12-10-01) "

Form: passive NP adjunct
Passive Subject: aircraft
PP ("to"): 200 pounds in tanks

" (a , d) **refuel** aircraft until fwd fuel low and aft fuel low caution lights go out . "

Form: imperative
Object: aircraft

7. impregnate

" if fuel spillage occurs on surface of aircraft , area shall be checked to determine if fuel has **impregnated** insulating blankets or duct insulation . "

Form: Perfect, "if"-S
Subject: fuel
Object: blankets, insulation

8. install

" technician a removes and installs valve (access cover 5403 or 6404) . "

Form: bare present
Subject: technician
Object: valve

" apply sealing compound and loosely **install** three screws , three washers , and three nuts in side of doorstop (two places) . "

Form: imperative
Object: screws, washers, nuts
PP ("in"): side of doorstop
AdvP: loosely

" flat washer shall be **installed** under head of bolt followed by sealing washer . "

Form: "shall" passive
Passive Subject: washer
PP ("under"): head of bolt

" one flat washer shall be **installed** between bolthead and sealing washer "

Form: "shall" passive
Passive Subject: washer
PP ("between"): bolthead and washer

" one flat washer shall be **installed** beneath nut on all four bolts . "

Form: "shall" passive
Passive Subject: washer
PP ("beneath"): nut

" screw shall be **installed** in lower outboard corner of pump and support . "

Form: "shall" passive
Passive Subject: screw
PP ("in"): corner of pump and support

" thick flange shall be **installed** toward trailing edge of o-ring groove . "

Form: "shall" passive
Passive Subject: flange
PP ("toward"): edge of o-ring groove

" bolts shall be **installed** from aft side of bulkhead . "

Form: "shall" passive
Passive Subject: bolts
PP ("from"): side of bulkhead

" either of two valves may be **installed** , part number 68c-1ms or 8101001-1 . "

Form: "may" passive
Passive Subject: bolts

" steps 20 through 22 may be omitted if tube and engine feed line copuling can be easily **installed** . "

Form: "can" passive
Passive Subject: tube, coupling

" when **installing** valve , spanner wrench adapter shall not be positioned in helicoil inserts "

Form: "when" gerund
Object: valve

" if installing check valve , omit step 4 . "

Form: "if" gerund
Object: valve

" if only regulator is to be **installed** , omit steps 3 through 7 . "

Form: "is to" passive
Passive Subject: regulator

" if right wing strainer and/or valve is being **installed** , omit steps 18.1 and 18.2 . "

Form: progressive passive

Passive Subject: strainer, valve

" verify lamp being **installed** is free of contamination . "

Form: gerund passive NP adjunct
Passive Subject: lamp

" pylon fuel air disconnect valve is **installed** against approximately 32 pounds spring pressure . "

Form: passive
Passive Subject: valve
PP ("against"): pressure

" lubricate and **install** seal (da4536-32) , washer , and bolt (nas6204-4) in lower inboard corner of upper inlet tube . "

Form: imperative
Object: seal, washer, bolt
PP ("in"): corner of tube

" lubricate and **install** packing (m25988-1-008) on indicator assembly . "

Form: imperative
Object: packing
PP ("on"): assembly

" position nipple in bracket and loosely **install** nut . "

Form: imperative
Object: nut
AdvP: loosely

" _44f1 to **install** actuator , it will be necessary to engage splines with actuator rotated 90 degrees clockwise from its normal mounting position and then lower and rotate actuator counterclockwise to mounting position . "

Form: "to"-S
Object: actuator

" plug is **installed** by pushing in and turning clockwise one-third turn . "

Form: passive
Passive Subject: plug

" position tube support and pressurization tube clamps and **install** bolt securing vent tube to pressurization tube . "

Form: imperative
Object: bolt

" coupling remover shall be **installed** in open position (lever all the way forward) . "

Form: "shall" passive

Passive Subject: coupling remover
PP ("in"): open position

" **install** two nuts . "

Form: imperative
Object: nuts

" lubricate four packings (m25988-1-214) and **install** on JFS bypass fuel tube connections . "

Form: imperative
Object: packings (zero)
PP ("on"): connections

" lubricate packing (m25988-1-904) and **install** on union . "

Form: imperative
Object: packing (zero)
PP ("on"): union

" (2) **install** aerial refuel slipway door (jg28-21-04) . "

Form: imperative
Object: door

" (2) **install** aircraft centerline tank selective refueling fuel shutoff valve (jg28-24-09) . "

Form: imperative
Object: valve

" (2) **install** access cover 5313 or 6314 using three screws . "

Form: imperative
Object: access cover

" **install** bolt with head on right side , two washers , and nut . "

Form: imperative
Object: bolt, washers, nut
PP ("with"): head on right side

" **install** shoulder bolt , two washers , and nut . "

Form: imperative
Object: bolt, washers nut

" align pressurization tube , position two sleeves , and **install** two couplings . "

Form: imperative
Object: couplings

" **install** access panel 3310 using two bolts , two washers , and two nuts . "

Form: imperative

Object: access panel

" **install** 18 bolts flush with surface of protective-frame . "

Form: imperative
Object: bolts
AdvP: flush with surface

" **install** cotter pin . "

Form: imperative
Object: pin

" **install** safety wire . "

Form: imperative
Object: wire

" **install** switch . "

Form: imperative
Object: switch

" **install** tee . "

Form: imperative
Object: tee

" (2,3) **install** forward right main glareshield or aft right main glareshield (jg53-00-22) . "

Form: imperative
Object: glareshield

" 4 . _44f5 (a) **install** three screws and bolt . "

Form: imperative
Object: screws, bolt

" **install** heat exchanger to tee fuel tube . "

Form: imperative
Object: heat exchanger
PP ("to"): tube

" **install** protective device on standpipe . "

Form: imperative
Object: device
PP ("on"): standpipe

" **install** retainer ring . "

Form: imperative
Object: ring

" **install** adg valve . "

Form: imperative
Object: valve

" **install** new lamp in lens . "

Form: imperative
Object: lamp
PP ("in"): lens

" **install** union in shutoff valve . "

Form: imperative
Object: union
PP ("in"): valve

" **install** cartridge in housing . "

Form: imperative
Object: cartridge
PP ("in"): housing

" **install** clamp on check valve . "

Form: imperative
Object: clamp
PP ("on"): valve

" **install** elbow on pump . "

Form: imperative
Object: elbow
PP ("on"): pump

" lubricate and **install** packing (m25988-1-008) on indicator assembly . "

Form: imperative
Object: packing
PP ("on"): assembly

" **install** check valve with flow direction arrow pointing forward . "

Form: imperative
Object: valve
PP ("with"): arrow pointing forward

9. lift

" pin is reset by **lifting** reset lever and pushing indicator pin into housing . "

Form: "by" gerund
Object: lever

" to manually open slipway door assembly , **lift** at hinge between forward and aft doors and fold aft door under forward door ; "

Form: imperative
Object: door assembly (zero)
PP ("at"): hinge

" raise slipway door assembly by **lifting** at hinge between forward and aft doors , fold aft door under forward door , "

Form: "by" gerund
Object: door assembly (zero)
PP ("at"): hinge

" **lift** panel and disconnect electrical connector . "

Form: imperative
Object: panel

10. line

" missing tooth on actuator shaft will **line** up with actuator indicator in a properly assembled actuator . "

Form: "will"
Particle: up
Subject: tooth
PP ("with"): indicator

11. place

" all support equipment required for refueling and fuel system maintenance shall be **placed** at maximum distance from aircraft that hoses and-or cables allow . "

Form: "shall" passive
Passive Subject: equipment
PP ("at"): maximum distance

" remaining washers (if any) shall be **placed** under nut . "

Form: "shall" passive
Passive Subject: washers
PP ("under"): nut

" lubricate grommet and **place** on test set vacuum adapter . "

Form: imperative
Object: grommet (zero)
PP ("on"): adapter

" if electrical operation failed to close valve , **place** valve actuator indicator in full closed (inboard) position . "

Form: imperative
Object: indicator
PP ("in"): closed position

12. position

" failure to **position** aft bracket properly on shoulder bolt may inhibit movement of fuel manifolds connected to engine and overstress manifold seal resulting in fuel leaks . "

Form: "to"-S
Subject: pro-arb?
Object: bracket
AdvP: properly
PP ("on"): shoulder bolt

" bracket shall be **positioned** so that slot in bracket aligns with alignment tab on tube . "

Form: "shall" passive
Passive Subject: bracket

" packing shall be **positioned** to side of o-ring groove which enters aerial refuel receptacle bore first so that slipper seal angles down toward inserting end of sleeve assembly . "

Form: "shall" passive
Passive Subject: packing
PP ("to side of"): groove

" when installing valve , spanner wrench adapter shall not be **positioned** in helicoil inserts or damage to inserts will result . "

Form: "shall" passive, neg
Passive Subject: wrench adapter
PP ("in"): helicoil inserts

" pump shall be **positioned** as high on support as possible and support position maintained until bolts and screw have been torqued . "

Form: "shall" passive
Passive Subject: pump
PP ("on"): support (as high as possible)

" retainer ring shall be **positioned** with small flange on inner surface mounted upward . "

Form: "shall" passive
Passive Subject: ring
PP ("with"): flange, mounted upward

" when power switch is **positioned** to on , only ready light should come on . "

Form: passive, "when"-S
Passive Subject: switch
PP ("to"): on

" (b) engine fuel shutoff valve actuator indicator moves smoothly and without slowing or oscillating to full open (outboard) position within 4 seconds after master switch is **positioned** . "

Form: passive, "after"-S
Passive Subject: switch

" (a , b) adjust two jamnuts as required to **position** switch so plunger touches slipway door assembly . "

Form: "to"-S
Subject: agent
Object: switch

" 2 . _66fd carefully slide matrix assembly forward to disengage two bushings ; then **position** matrix assembly to permit access to amplifier . "

Form: imperative
Object: matrix assembly

" **position** master switch guard down to guarded position . "

Form: imperative
Object: switch guard
AdvP: down
PP ("to"): guarded position

13. pump

" **pump** handle on vacuum pump to apply 5 psig vacuum as indicated on test set gage . "

Form: imperative
Object: handle

14. push

" to manually open slipway door assembly , lift at hinge between forward and aft doors and fold aft door under forward door ; then **push** slipway door assembly down to full open position . "

Form: imperative
Object: door assembly
AdvP: down
PP ("to"): open position

" pin is reset by lifting reset lever and **pushing** indicator pin into housing . "

Form: "by" gerund
Object: pin
PP ("into"): housing

" plug is removed by **pushing** in and turning counterclockwise one-third turn . "

Form: "by" gerund
Particle: in
Object: plug (zero)

" do not **push** inboard on engine side of serrated locknut as coupling nut is turned or an overtight condition may occur , causing difficulty disconnecting quick-disconnect . "

Form: imperative, neg
AdvP: inboard
PP ("on"): engine side of locknut

" **push** down on forward door near hinge until slipway door assembly is fully closed . "

Form: imperative
AdvP: down
PP ("on"): door

" if protrusion exceeds 0.75 inch max , **push** drive wire back into coupling nut until an approximate 0.25-inch nominal protrusion exists . "

Form: imperative
Object: wire
AdvP: back
PP ("into"): nut

" **push** poppet in fuel disconnect valve upward and drain residual fuel . "

Form: imperative
Object: poppet
AdvP: upward

" using drain tool , **push** in on poppet and release to verify operation . "

Form: imperative
Particle: in
PP ("on"): poppet

" using drain tool , **push** poppet in and rotate approximately one-half turn counterclockwise until marks align . "

Form: imperative
Particle: in
Object: poppet

15. raise

" **raise** slipway door assembly by lifting at hinge between forward and aft door . "

Form: imperative
Object: door assembly

" **raise** slipway door assembly higher by moving up and aft . "

Form: imperative
Object: door assembly
AdvP: higher

16. rest

" to avoid damaging aircraft , insure hydraulic hoses are on proper connections and wire bundle splice does not **rest** on bulkhead segment under air refuel receptacle . "

Form: S-comp, neg
Subject: splice
PP ("on"): bulkhead segment

" if aft end of forward door **rests** above contour of access panel 3437 by more than 0.15 inch , lengthen actuator by turning rod end counterclockwise . "

Form: "if"-S
Subject: end of door
PP ("above . . . by more than"): contour

17. saturate

" clean 21 bolts and 6 screws using clean cheesecloth **saturated** with solvent compound . "

Form: passive NP adjunct
Passive Subject: cheesecloth
PP ("with"): solvent compound

18. smear

" avoid **smearing** compound grease on painted surface ; it is difficult to remove and can n't be painted . "

Form: "avoid" gerund
Subject: agent (zero)
Object: grease
PP ("on"): surface

19. spill

" to avoid fire and explosive hazards , **spilled** fuel shall be cleaned up immediately . "

Form: passive NP adjunct
Passive Subject: fuel

20. spray

" **spray** a small amount of lubricant (mil-c-87177a) into actuator connector and electrical connector (2822p1) . "

Form: imperative
Object: lubricant
PP ("into"): connector

21. stick

" piston of valve shall be free of any contaminants or foreign substance which could cause piston to **stick** open , creating a fuel venting problem or causing damage to valve . "

Form: "to"-S
Subject: piston (zero)
AdvP: open

22. tag

" hydraulic hoses shall be **tagged** for identification . "

Form: "shall" passive
Passive Subject: hoses

" cut five wires , staggering cuts , and **tag** . "

Form: imperative
Object: wires (zero)

23. trap

" fuel may be **trapped** in fuel pump or engine feed line assembly ; "

Form: "may" passive
Passive Subject: fuel
PP ("in"): pump, feed line assembly

24. wash

" if fuel splashes into eyes , **wash** eyes immediately ; then seek medical services . "

Form: imperative
Object: eyes

25. wipe

" after aerial refuel receptacle has been **wiped** dry , fuel leakage at poppet or receptacle weep hole shall not exceed one drop in 5 minutes . "

Form: passive, perfect
Passive Subject: receptacle
AdvP: dry

" **wipe** connector clean of leak detection compound using cheesecloth . "

Form: imperative
Object: connector
AdvP: clean (of . . .)

" **wipe** 21 bolts and 6 screws dry using clean cheesecloth . "

Form: imperative
Object: bolts, screws
AdvP: dry

" using cheesecloth , **wipe** any residual fuel from transfer tube area . "

Form: imperative
Object: fuel
PP ("from"): tube area

" **wipe** off any excess sealing compound from strainer mounting flange using clean cheesecloth . "

Form: imperative
Particle: off
Object: sealing compound
PP ("from"): flange

10. References

App85 Douglas Appelt. Planning English Sentences. Cambridge University Press, Cambridge England, 1985.

Bad75 Norman I. Badler. Temporal Scene Analysis: Conceptual descriptions of object movements. PhD thesis, University of Toronto, 1975.

BALC95 Hanêne Ben-Abdallah, Insup Lee, and Jin-Young Choi. GCSR: a Graphical Language with Algebraic Semantics for the Specification of Real-Time Systems. Technical Report MS-CIS-95-09, Dept. of Computer and Information Science, Univ. of Pennsylvania, 1995.

BBN94 BBN Inc., Cambridge, MA 02138. OMAR User/Programmer Manual, Version 1.0, December 1994. BBN Report No. 7997 (1).

BKA95 H. Bekkering, H. Kigma, H. Adams, A. Van der Aarssen, and H Whiting. Interference between saccadic eye and goal directed hand movements. Experimental Brain Research, 106:475--484, 1995.

BL80 I. Bartenieff and D. Lewis. Body Movement: Coping with the Environment. Gordon and Breach Science Publishers, New York, 1980.

BPW93 N. I. Badler, C. W. Phillips, and B. L. Webber. Simulating Humans: Computer Graphics Animation and Control. Oxford University Press, New York, NY, 1993.

BWKE91 N. I. Badler, B. L. Webber, J. Kalita, and J. Esakov. Animation from instructions. In N. Badler, B. Barsky, and D. Zeltzer, editors, Making Them Move: Mechanics, Control, and Animation of Articulated Figures, pages 51--93. Morgan-Kaufmann, San Mateo, CA, 1991.

Caw92 Alison Cawsey. Explanation and Interaction: The Computer Generation of Explanatory Dialogues. MIT press, 1992.

Coh84 Philip Cohen. The pragmatics of referring and the modality of communication. Computational Linguistics, 10(2):97--146, April-June 1984.

Coh87 Robin Cohen. Analyzing the structure of argumentative discourse. Computational Linguistics, 13(1-2):11--24, 1987.

DH91 Robert Dale and Nicholas Haddock. Content determination in the generation of referring expressions. Computational Intelligence, 7(4):252--265, 1991.

EPM93 Henrik Eriksson, Angel R. Puerta, and Mark A. Musen. Generation of knowledge-acquisition tools from domain ontologies. Technical Report KSL-93-56, Stanford University, Stanford, CA, 1993.

GS86 Barbara Grosz and Candace Sidner. Attention, intentions, and the structure of discourse. Computational Linguistics, 12:175--204, 1986.

Hov88 Eduard H. Hovy. Planning coherent multisentential text. In ACL, pages 163--169, 1988.

KKR91 Richard Kittredge, Tanya Korelsky, and Owen Rambow. On the need for domain communication knowledge. Computational Intelligence, 7(4):305--314, 1991.

KL96 Jugal K. Kalita and Joel Lee. An informatl semantic analysis of motion verbs based on physical primitives. Computational Intelligence, 1996.

KMB96 E. Kokkevis, D. Metaxas, and N. Badler. User-controlled physics-based animation for articulated figures. In Computer Animation. IEEE Press, 1996.

LP95 James C. Lester and Bruce W. Porter. Developing and empirically evaluating robust explanation generators: the KNIGHT experiments, 1995. Journal submission.

McK85 Kathleen R. McKeown. Text generation. Using discourse strategies and focus constraints to generate natural language text. Cambridge University Press, 1985.

MFCS87 Mark A. Musen, Lawrence M. Fagan, David M. Combs, and Edward H. Shortliffe. Use of a domain model to drive an interactive knowledge-editing tool. International Journal of Man-Machine Studies, 26:105--121, 1987.

Moo95 Johanna D. Moore. Participating in Explanatory Dialogues. MIT press, 1995.

MP93 Johanna D. Moore and Cécile L. Paris. Planning text for advisory dialogues: Capturing intentional and rhetorical information. Computational Linguistics, 19(4):651--695, 1993.

Pal90 M. Palmer. Semantic Processing for Finite Domains. Cambridge University Press, Cambridge, England, 1990.

PS93 Scott Prevost and Mark Steedman. Generating contextually appropriate intonation. In Proceedings of the Sixth Conference of the European Chapter of ACL, pages 332--340, Utrecht, 1993.

RD92 Ehud Reiter and Robert Dale. A fast algorithm for the generation of referring expressions. In Proceedings of COLING, pages 232--238, 1992.

Rei91 Ehud Reiter. A new model of lexical choice for nouns. Computational Intelligence, 7(4):240--251, 1991.

Rei94 Ehud Reiter. Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In Seventh International Workshop on Natural Language Generation, pages 163--170, June 1994.

Rei97 Barry D. Reich. An Architecture for Behavioral Locomotion. PhD thesis, University of Pennsylvania, 1997.

SC88 P. Suppes and C. Crangle. Context-Fixing Semantics for the Language of Action. In J. Dancy, J. Moravcsik, and C. Taylor, editors, Human Agency: Language, Duty, and Value, pages 47--76. Stanford University Press, Stanford, CA, 1988.

SD96 Matthew Stone and Christine Doran. Paying heed to collocations. In International Workshop on Natural Language Generation, 1996.

Sib92 Penelope Sibun. Generating Text without Trees. Computational Intelligence: Special Issue on Natural Language Generation, 8(1), 1992.

SvNPM90 Stuart Shieber, Gertjan van Noord, Fernando Pereira, and Robert Moore. Semantic-head-driven generation. Computational Linguistics, 16:30--42, 1990.

VM95 Keith Vander Linden and James Martin. Expressing Local Rhetorical Relations in Instructional Text. Computational Linguistics, 21(1):29--57, 1995.

WAB91 Wolfgang Wahlster, Elisabeth André, Son Bandyopadhyay, Winfried Graf, and Thomas Rist. WIP: The coordinated generation of multimodal presentations from a common representation. In Oliviero Stock, John Slack, and Andrew Ortony, editors, Computational Theories of Communication and their Applications. Berlin: Springer Verlag, 1991.

WBB92 Bonnie Webber, Norman Badler, F. Breckenridge Baldwin, Welton Becket, Barbara Di Eugenio, Christopher Geib, Moon Jung, Libby Levison, Michael Moore, and Michael White. Doing What You're Told: Following Task Instructions In Changing, but Hospitable Environments. Technical Report MS-CIS-92-74, University of Pennsylvania, Philadelphia, PA, 1992.

WBE95 B. Webber, N. Badler, B. Di Eugenio, C. Geib, L. Levison, and M. Moore. Instructions, intentions and expectations. Artificial Intelligence Journal, 73:253--269, 1995.

WD90 Bonnie Webber and Barbara Di Eugenio. Free Adjuncts in Natural Language Instructions. In Proceedings of COLING, pages 395--400, 1990.

YM94 R. Michael Young and Johanna D. Moore. DPOCL: A Principled Approach to Discourse Planning. In Seventh International Workshop on Natural Language Generation, pages 13--20, Kennebunkport, Maine, 1994.

YMVS91 Gijoo Yang, Kathleen F. McCoy, and K. Vijay-Shanker. From functional specification to syntactic structures: systemic grammar and tree-adjoining grammar. Computational Intelligence, 7(4):207--219, 1991.

6 Appendix B: Storyboards and Design Notes for DEPTH LSAR Database Interface

Storyboards and Design Notes
for
DEPTH LSAR Database Interface

T. L. Mann
Hughes Missile Systems Company
P.O. Box 11337
Tucson, AZ 85734-1337

April 7, 1998

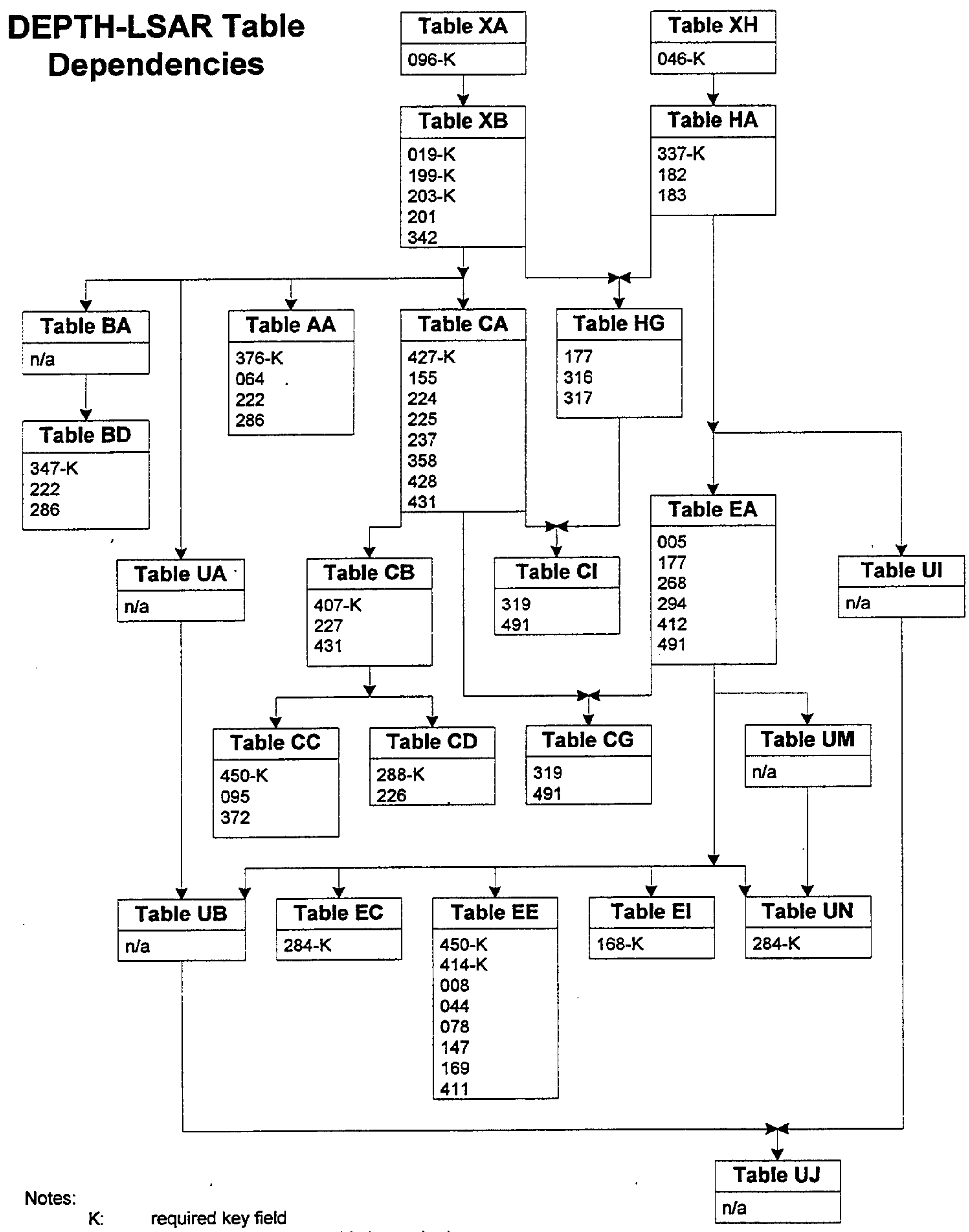
Table of Contents

List of Figures

List of Tables

1. Introduction

This document establishes a technical description of the DEPTH LSAR Database Interface.



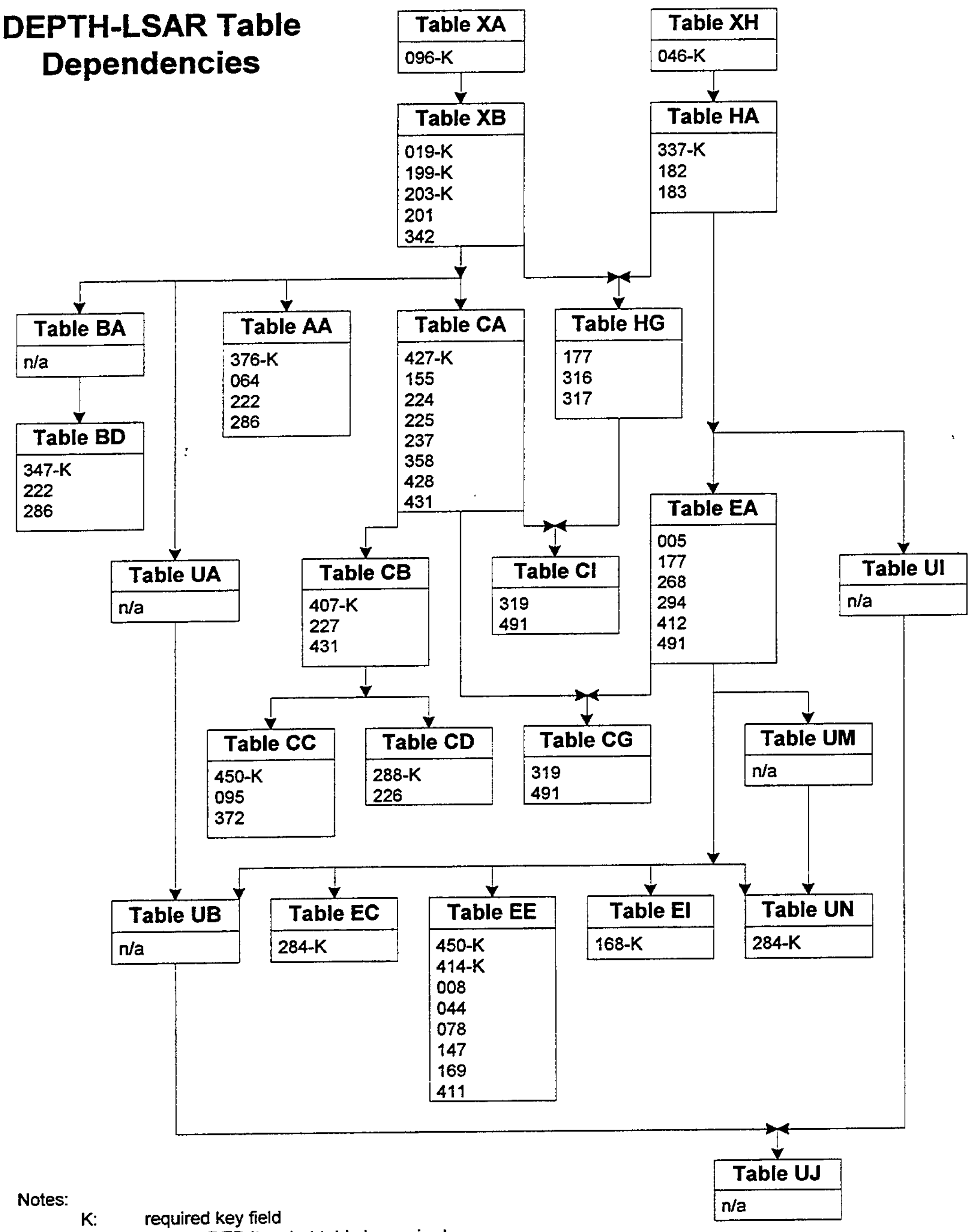
Notes:
K: required key field
n/a: no new DED item but table is required

Figure 1shows the dependencies that exist between the tables used by DEPTH and the DED items that DEPTH shall be able to update and those referred to as design goals. For convenience, these will be referred to as the table name followed by the DED number. For example, DED 096 in table XA will be referred to as XA096.

To make an entry or update in a table, the parent table must have an entry and the key fields must be completed. For example, to make an entry in table BD requires an entry in tables XA, XB, and BA. In addition, the key fields in table BD must be completed. The key fields required are the union of those in the parent tables, known as foreign keys, and any local ones, known as primary keys. For BD these are foreign keys XA096, XB019, XB199, and XB203, and primary key BD347.

There are some tables that require special consideration. As shown in

DEPTH-LSAR Table Dependencies

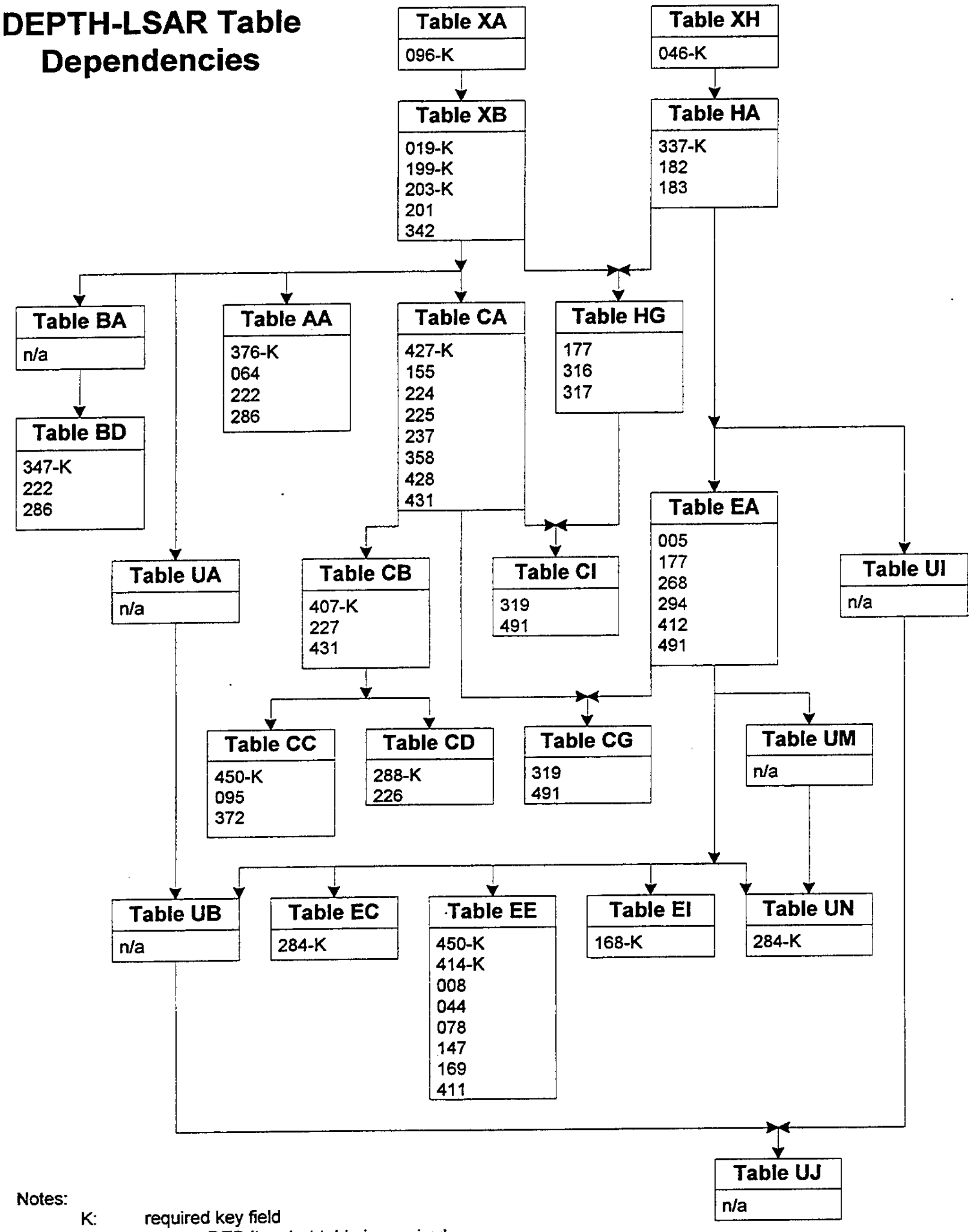


Notes:
- K: required key field
- n/a: no new DED item but table is required

Figure 1, tables CI, UJ, and UN have two paths that converge to a single parent table. Each path is for different types of information so the key fields will contain different information except for table XA. Therefore, the required foreign keys include two keys for each key in the single parent table, one set for each path. For example, to make an entry or update in tables UN and UJ requires double entries for the key fields found in tables XH and HA, one from each path. Table CI requires double entries for the key fields found in table XB. Only a single entry is required for the field in XA since it is the same regardless of the path.

If an entry is made for a piece of support equipment (SE) that requires an adapter, the entry in table EA will link the SE to an adapter documented in tables UI and UJ. If the adapter has not been entered first, you will not be allowed to enter the information for the SE.

DEPTH-LSAR Table Dependencies



Notes:

K: required key field

n/a: no new DED item but table is required

Figure 1: Dependencies of LSAR Tables Used by DEPTH

2. Flow Overview of New and Modified Dialog Boxes

All of the LSAR dialog entries shall be grayed out until LSAR is enabled. In workspace setup there is a button to enable LSAR. When the button is selected, a logon dialog shall open. Once the user is logged on to the LSAR server the LSAR dialog entries shall be enabled beginning with the rest of the LSAR setup dialog entries. If the workspace is closed with LSAR enabled, when loaded again, DEPTH shall open a logon dialog to prompt the user to enter a password to continue using LSAR.

If the logon fails or if the connection to the server is dropped and DEPTH is not able to restore the connection, LSAR shall be disabled. If the user wishes to change LSAR servers, LSAR must first be disabled and then enabled from the workspace setup dialog.

Once enabled (logged on), LSAR information can be entered in the corresponding dialog fields for fasteners and inserted objects. DEPTH must be logged on to the server since much of the information shall be maintained in the LSAR database rather than in the DEPTH workspace. This is required to accommodate changes that may be made to the LSAR database outside of DEPTH. If enabled in the middle of a session, for existing objects, tools, and fasteners to be included in the LSAR database, the user will need to input the required information in the modify dialog box for each of these figures.

When entering simulation mode, before a simulation step can be defined, the user shall be required to complete the LSAR setup dialog where the task, subtask, and task element are defined. This shall be the first step in the simulation. The user shall then be allowed to setup other simulation steps. At the end of a task, subtask, or task element the user is expected to again enter an LSAR setup step. During a simulation, when an LSAR step is encountered, DEPTH shall update the LSAR database with the information collected during simulation related to the respective task, subtask, and task element.

The LSAR database shall only be updated during a simulation run when requested by the user. When making an LSAR run, movie making shall be disabled. A checkbox is provided on the run simulation dialog for this purpose. This allows the user to run various scenarios without cluttering the LSAR database with the preliminary information. If the connection to the LSAR database is broken during simulation, the run will be aborted and DEPTH shall attempt to log back on to the server. If DEPTH is able to restore the server connection, the user shall be able to run the simulation again.

When a task simulation has successfully completed a post simulation dialog shall be opened for the user to add information related to observations made during the simulation.

If the user wants to use the same workspace to simulate another task, the LSA control number information must be changed on the DWS Setup dialog. This shall cause DEPTH to reinitialize the LSAR information contained in the respective members of the C++ objects. The user shall again be required to use the modify figure dialog to input the information for each figure into the LSAR database. A design goal will be to have DEPTH update the database with all the existing figure information when the LCN information is changed.

3. Dialog Box Details

Detailed information about new and modified dialog boxes used to implement the LSAR function in DEPTH are covered in Table 2: Details for New and Modified Dialog BoxesTable 2. In some cases, where the dialog box does not change, only the information on how the DED items are used is included without showing a figure of the dialog. For modified dialogs, only the details of the modifications are described. There are no details about existing function unless it has a direct application to LSAR such as the name of a figure. Figures of dialog boxes are not exact representations of how they shall appear in DEPTH. They are used strictly to highlight the appearance of the added LSAR sections. Consequently, peripheral figure type radio buttons and figure selection buttons may be missing. Table 1 provides a description of the headings used in Table 2.

Heading	Description
DED	LSAR data element definition number as defined in Mil-Std-1388-2B
TABLE	LSAR table code as defined in Mil-Std-1388-2B
SIZE	Size of the field required in the LSAR database as defined in Mil-Std-1388-2B
DIALOG	DEPTH dialog box that utilizes the value of this DED item
WIDGET	Type of widget used to collect information from the user. A "---" indicates a value is used without user input.
K	Key field for the table as defined in Mil-Std-1388-2B
O	Table entry is an optional work item otherwise known as a design goal.
M	Value shall be retained in memory as a member of a C++ object
F	Value shall be retained with the workspace file
COMMENTS	Description of the table, DED, and actions performed.

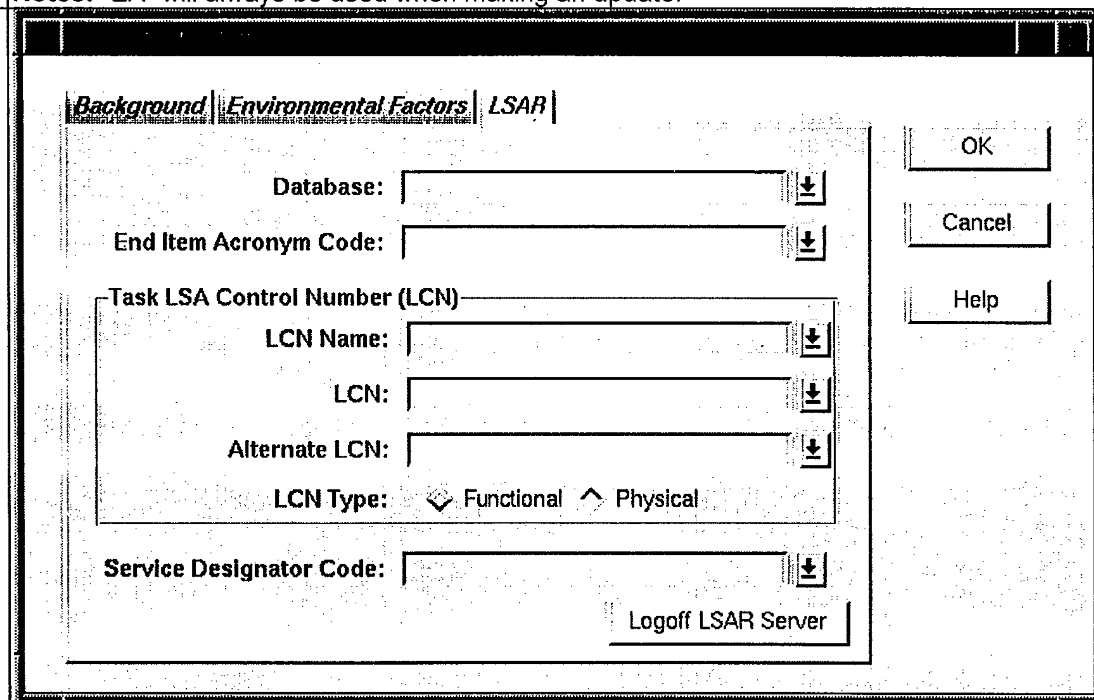
Table 1: Definition of headings

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
	aa								Operations and Maintenance Requirement Identifies operations requirements for new system/equipment.
	bd								Reliability, Availability, and Maintainability Indicator Characteristics Contains reliability and maintainability characteristics of item under analysis
	ca								Task Requirements Contains task level, personnel, and training information.
	cb								Subtask Requirement Contains subtask related information. References to other subtasks can also be made.
	cc								Sequential Subtask Description Contains subtask descriptions that are associated with a given task. This includes any notes, cautions, or warnings.
	cd								Subtask Personnel Requirements Contains information about the personnel and support requirements for each subtask.

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
	cg								Task Support Equipment Relates information needed for a task to the SE
	ci								Task Provisioned Item Links provisioned items to a task. Documents spare and repair parts needed to support the task.
	ea								Support Equipment Contains information about SE
	ec								Support Equipment Parameters Documents the parameters that can be measured, generated, etc. by the SE. These are used to compare with UUT parameters to determine if the SE is adequate.
	ee								Support Equipment Narrative Documents different types of narrative text for SE.
	ei								Input Power Source Used to document the power requirements of the equipment.
	ha								Item Identification Contains parts information.
	hg								Part Application Provisioning Information related to parts in a specific hardware application. This can only be used if the physical parts exist.
	un								Support Equipment Unit Under Test Parameter Group Documents the parameters measured on the UUT and associates them with the corresponding SE making the measurement.
	xa								End Item Acronym Code Code used to define the LSAR system being documented
	xb								LSA Control Number Indentured Item Contains all LCN's and information about the indentured location of the LCN
	xh								Commercial and Government Entity Code Contains all the cage codes and addresses.

			Delete Figure	Button					**Title: OK**
									Item: OK button to remove a figure
									Default: n/a
									Dependency: n/a
									Action:
									If fastener or object
									• If HA337 has value in DEPTH
									  • If end item part
									    • Decrement HG316 and HG317
									  • If quantity is one

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									• Remove from database • If EA005 is "Y" • Remove adapter information If human • If CD288 id has value in DEPTH • Delete from LSAR database If tool • If HA337 has value in DEPTH • Decrement CG319 and CI319 **Notes:** n/a
288	cd	3	Delete Figure	---	X		X	X	**Item:** This is the id for a human. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Delete from LSAR database
316	hg	4	Delete Figure	---					**Item:** Quantity of part used per assembly. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Decrement value when delete
317	hg	5	Delete Figure	---					**Item:** Quantity of part used in system. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Decrement value when delete
319	cg	5	Delete Figure	---					**Item:** Number of items used to perform the task **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Decrement value when delete
319	ci	5	Delete Figure	---					**Item:** Number of parts used to perform the task **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Decrement value when delete
491	cg	2	Delete Figure	---					**Item:** This is the unit of measure for the quantity of SE given by CG319 **Default:** n/a

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Dependency: n/a **Action:** n/a **Notes:** "EA" will always be used when making an update.
491	ci	2	Delete Figure	---					**Item:** This is the unit of measure for the quantity of parts given by CI319 **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** "EA" will always be used when making an update.
			DWS Setup	Window					 **Figure 3: DWS Setup**
			DWS Setup	tixComboBox			X	X	**Title: Database** **Item:** Name of the LSAR database to access. **Default:** LSAR object value else first in list of those accessible **Dependency:** Server

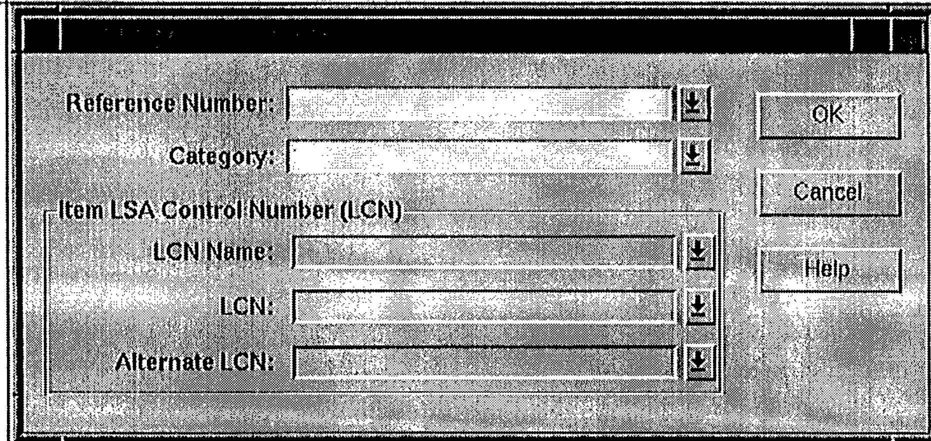
DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Action: • Query database for XA096 list. • If value in LSAR object is in list, move to top of list • Set XA096 widget to value at top of list **Notes:** Must be authorized user to select database.

			DWS Setup	Button					**Title: OK**
									Item: OK button
									Default: n/a
									Dependency: n/a
									Action:
									• Save state of LSAR enable flag
									• If LSAR enabled
									    • If server, database, XA096, XB199, XB019, or XB203 changed, replace LSAR object with new one (reinitialized) with values selected in dialog
									    • Query XH046 and update if "DEPTH" is not in list
									    • Query XB203 and update if value is not in list
									    • Update BA so entries can be made to BD
									    • Save logon userid, server, and password in DEPTH
									Notes: n/a

			DWS Setup	Button					**Title: Cancel**
									Item: Cancel button
									Default: n/a
									Dependency: n/a
									Action:
									• Logged off server changed:
									    • Logoff server
									    • Reset flag to gray LSAR
									• Logged on server changed:
									    • Reset userid and server values from LSAR object
									    • Logon to server using password in LSAR object. On fail post error and reset LSAR enable flag
									    • Set LSAR flag
									• Server changed:
									    • Logoff server
									    • Reset userid and server values from LSAR object
									    • Logon to server using password in LSAR object. On fail post error and reset LSAR enable flag

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									• Set LSAR flag **Notes:** Select or deselect of LSAR enable will handle the logon/logoff actions
			DWS Setup	Button			X	X	**Title: Logoff LSAR Server** **Item:** Toggle button between logon and logoff of the LSAR Server **Default:** n/a **Dependency:** n/a **Action:** For "Logon LSAR Server": • Set userid to LSAR object value else UNIX userid • Set server to LSAR object value else first in list of those accessible • Open logon dialog. For "Logoff LSAR Server": • Logoff LSAR server and set flag to gray LSAR fields **Notes:** Set button text and actions based on status of LSAR enable flag
019	xb	2	DWS Setup	tixComboBox	X		X	X	**Title: Alternate LCN** **Item:** Alternate logistics control code **Default:** LSAR object value else first in list from database **Dependency:** XB199 value **Action:** • Query database for XB203 list • If LSAR object value is in list, move to top of list • Set DWS Setup XB203 widget to value at top of list **Notes:** Range is 00-99 but is not sequential.
046	xh	5	DWS Setup	---	X				**Item:** This is the cage code **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Input "DEPTH" in this field. When the analyst sees this invalid cage code it will act as a flag that this row was updated by DEPTH. The analyst can then evaluate the changes that were made.
096	xa	10	DWS Setup	tixComboBox	X		X	X	**Title: End Item Acronym Code** **Item:** End item acronym code that uniquely identifies the system (e.g. TOW, Sparrow) **Default:** LSAR object value else first in list from database **Dependency:** Server **Action:** • Query database for XB201 list. • If value based on key values in LSAR object is in list, move to top of list • Set XB201 widget to value at top of list

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Notes: n/a
199	xb	18	DWS Setup	tixComboBox	X		X	X	**Title: LCN** **Item:** Logistics support analysis control number that represents a functional or hardware breakdow of the system. **Default:** LSAR object value else first in list from database **Dependency:** XB201 value **Action:** • Query database for XB019 list • If LSAR object value is in list, move to top of list • Set XB019 widget to value at top of list **Notes:** Generally the selection list will only be 1 item.
201	xb	19	DWS Setup	tixComboBox					**Title: LCN Name** **Item:** Name associated with the logistics control number **Default:** Value based on key values in LSAR object else first in list from database **Dependency:** XA096 value **Action:** • Query database for XB199 list • If LSAR object value is in list, move to top of list • Set DWS Setup XB199 widget to value at top of list **Notes:** Generally this will be 1 item.
203	xb	1	DWS Setup	Radio Button	X		X	X	**Title: LCN Type** **Item:** Indicates if the logistic control number represents either a physical or functional breakdown. Physical means the parts exist. Functional means the parts do not exist yet. **Default:** LSAR object value else functional **Dependency:** XB019 **Action:** • Query database for AA376 list • If LSAR object value is in list, move to top of list • Set AA376 widget to value at top of list **Notes:** Used whenever a row in the table is being updated. User can change value which may result in a new row being entered in XB
376	aa	1	DWS Setup	tixComboBox	X		X	X	**Title: Service Designator Code** **Item:** Service designator code identifying the military or agency in charge. **Default:** LSAR object value else first in list from database **Dependency:** XB203 value **Action:** n/a **Notes:** This information is required to make an entry in AA. It is entered when the database is setup

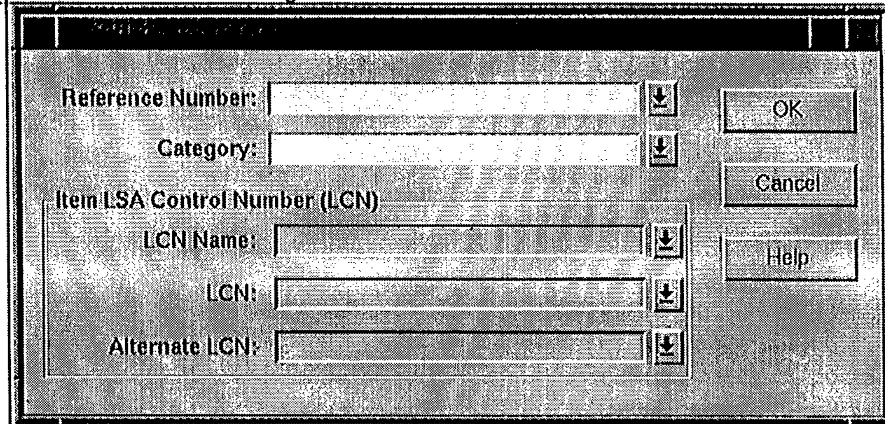
DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									initially outside of DEPTH. The code will be translated to text for user selection.
			Insert Fastener	Window					 **Figure 4: Insert Fastener**

			Insert Fastener	Button					**Title: OK**
									Item: OK button
									Default: n/a
									Dependency: n/a
									Action:
									If HA337 has value
									• Update HA337
									• Update HA182 only if this is the first time the HA337 reference has been used.
									• Update HA183 only if this is the first time the HA337 reference has been used
									• Update XB203 with "P"
									• If end item part is yes
									    • Query HG316 and HG317 from LSAR
									    • Count those in fastener ring having the same HA337 value
									    • Update HG317 = (HG317 query) - (HG316 query) + count
									    • Update HG316 with this count
									Notes: n/a

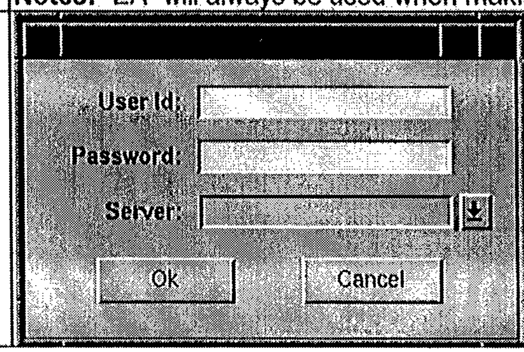
019	xb	2	Insert Fastener	tixComboBox	X		X	X	**Title: Alternate LCN** **Item:** Alternate logistics control code **Default:** LSAR object value else first in list from database **Dependency:** XB199 value

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Action: n/a **Notes:** Range is 00-99 but is not sequential.
182	ha	19	Insert Fastener	---		X			**Item:** Name of the part **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Use the DEPTH name of the fastener for the first entry. If more of the same HA337 reference are made, do not change the name.
183	ha	5	Insert Fastener	---		X			**Item:** Code for approved names **Default:** 77777 **Dependency:** n/a **Action:** n/a **Notes:** This code is used with DED 182. The default is the code for a name that is not standard.
199	xb	18	Insert Fastener	tixComboBox	X		X	X	**Title: LCN** **Item:** Logistics support analysis control number that represents a functional or hardware breakdow of the system. **Default:** LSAR object value else first in list from database **Dependency:** XB201 value **Action:** • Query database for XB019 list • If LSAR object value is in list, move to top of list • Set XB019 widget to value at top of list **Notes:** Generally the selection list will only be 1 item.
201	xb	19	Insert Fastener	tixComboBox					**Title: LCN Name** **Item:** Name associated with the logistics control number **Default:** Value based on key values in LSAR object else first in list from database **Dependency:** XA096 value **Action:** • Query database for XB199 list • If LSAR object value is in list, move to top of list • Set Insert Fastener XB199 widget to value at top of list **Notes:** Generally this will be 1 item.
203	xb	1	Insert Fastener	---	X		X	X	**Item:** Indicates if the logistic control number represents either a physical or functional breakdown. Physical means the parts exist. Functional means the parts do not exist yet. **Default:** LSAR object value else functional **Dependency:** XB019 **Action:** n/a

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Notes: Always enter "P" in database. Assumes physical part exists if HA337 exists
316	hg	4	Insert Fastener	---					**Item:** Quantity of part used per assembly. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** The user selects the assembly from the LCN name list.
317	hg	5	Insert Fastener	---					**Item:** Quantity of part used in system. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** The user selects the assembly from the LCN name list.
337	ha	32	Insert Fastener	tixComboBox	X		X	X	**Title: Reference Number** **Item:** Reference number. This is normally the manufacturer's part number. **Default:** n/a **Dependency:** n/a **Action:** When not blank, ungray end item part radio buttons, XB201, XB199, and XB019 tixComboBoxes. **Notes:** Assume value indicates physical part which is required to make LSAR entry. Drop down shows those already in DEPTH.
hg	177	2	Insert Fastener	tixComboBox		X	X	X	**Title: Category** **Item:** Indicates how this part is used **Default:** No **Dependency:** n/a **Action:** n/a **Notes:** Determines which table to update. End item parts are entered in HG while others are entered in CI.
			Insert Human	Button					**Title: OK** **Item:** OK button to insert a human figure in the workspace **Default:** n/a **Dependency:** n/a **Action:** Query the database for the latest id. Index this by 1 for the new human. The id ranges from A to 999. **Notes:** n/a
288	cd	3	Insert Human	---	X		X	X	**Item:** This is the id for a human. **Default:** n/a **Dependency:** n/a **Action:** n/a

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Notes: When a person is generated, query the database for the latest id. Index this by 1 for the new human. The id ranges from A to 999.
			Insert Tool	Window					 **Figure 5: Insert Tool**
			Insert Tool	Button					**Title: OK** **Item:** OK button **Default:** n/a **Dependency:** n/a **Action:** Update HA337, HA182, HA183, EA177 Count number of tools in ring with same HA337 value Update CG319 and CI319 with count value **Notes:** Set button text and actions based on status of LSAR enable flag
019	xb	2	Insert Tool	tixComboBox	X		X	X	**Title: Alternate LCN** **Item:** Alternate logistics control code **Default:** LSAR object value else first in list from database **Dependency:** XB199 value **Action:** n/a **Notes:** Range is 00-99 but is not sequential.
177	hg	2	Insert Tool	tixComboBox		X			**Title: Category** **Item:** Identifies the type and category of an item using a code. **Default:** n/a **Dependency:** n/a **Action:** n/a

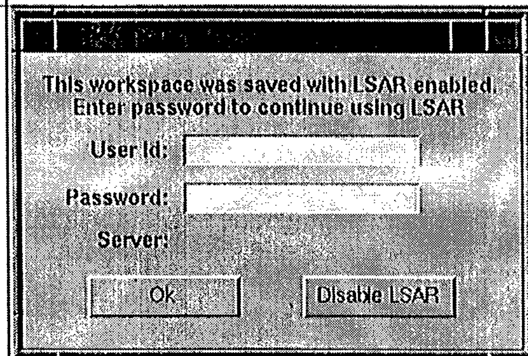
DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Notes: Uses the code for "common in DOD inventory"
182	ha	19	Insert Tool	---		X			**Item:** Name of the part **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** This value comes from toolbox file
183	ha	5	Insert Tool	---		X			**Item:** Code for approved names **Default:** 77777 **Dependency:** n/a **Action:** n/a **Notes:** This code is used with DED 182. This value will be 77777
199	xb	18	Insert Tool	tixComboBox	X		X	X	**Title: LCN** **Item:** Logistics support analysis control number that represents a functional or hardware breakdow of the system. **Default:** LSAR object value else first in list from database **Dependency:** XB201 value **Action:** • Query database for XB019 list • If LSAR object value is in list, move to top of list • Set XB019 widget to value at top of list **Notes:** Generally the selection list will only be 1 item.
201	xb	19	Insert Tool	tixComboBox					**Title: LCN Name** **Item:** Name associated with the logistics control number **Default:** Value based on key values in LSAR object else first in list from database **Dependency:** XA096 value **Action:** • Query database for XB199 list • If LSAR object value is in list, move to top of list • Set Insert Tool XB199 widget to value at top of list **Notes:** Generally this will be 1 item.
203	xb	1	Insert Tool	---	X		X	X	**Item:** Indicates if the logistic control number represents either a physical or functional breakdown. Physical means the parts exist. Functional means the parts do not exist yet. **Default:** LSAR object value else functional **Dependency:** XB019 **Action:** n/a **Notes:** Always enter "P" in database. Assumes physical part exists if HA337 exists

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
319	cg	5	Insert Tool	---					**Item:** Number of items used to perform the task **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Use a count of the tools having the same HA337 reference for this value
319	ci	5	Insert Tool	---					**Item:** Number of parts used to perform the task **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Use a count of the tools having the same HA337 reference for this value
337	ha	32	Insert Tool	tixComboBox	X		X	X	**Title: Reference Number** **Item:** Reference number. This is normally the manufacturer's part number. **Default:** n/a **Dependency:** n/a **Action:** n/a. **Notes:** This values comes from toolbox file. Gray other widgets if this has no value in it.
491	cg	2	Insert Tool	---					**Item:** This is the unit of measure for the quantity of SE given by CG319 **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** "EA" will always be used when making an update.
491	ci	2	Insert Tool	---					**Item:** This is the unit of measure for the quantity of parts given by CI319 **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** "EA" will always be used when making an update.
			Logon	Window					

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Figure 6: Logon
			Logon	tixComboBox			X	X	**Title: Server** **Item:** Name of the LSAR database server. **Default:** LSAR object value else first name on list of those accessible **Depend:** n/a **Action:** n/a **Notes:** Used in logon command when logon button is pressed.
			Logon	Entry			X	X	**Title: User Id** **Item:** Userid for the account on the LSAR database server **Default:** LSAR object value else UNIX userid **Dependency:** n/a **Action:** n/a **Notes:** Used in logon command when logon button is pressed.
			Logon	Entry			X		**Title: Password** **Item:** Server account password **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Data field will not show actual password entered. Plan is to show an '*' for each character entered. The password will be kept in memory but it will be broken into pieces so it can not be detected by reading memory directly. This is used in the logon command when logon button is pressed.

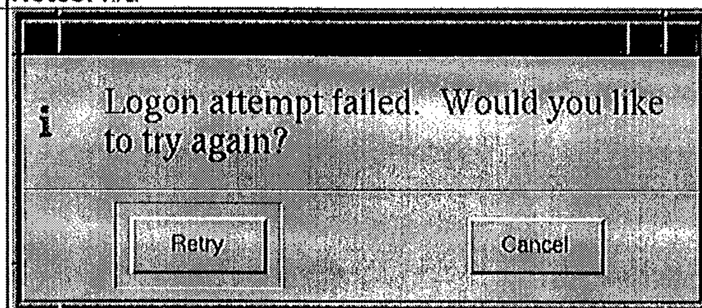
			Logon	Button					**Title: OK**
									Item: OK Button to send logon command
									Default: n/a
									Dependency: Userid, server name, and password fields must have values entered.
									Action: Send logon command to the server.
									• Pass:
									  • Query server for database list
									  • If database LSAR object value is in list, move it to top of list
									  • Set database widget to value at top of list
									  • Change setup button text to "Logoff LSAR Server"
									  • Set flag to enable LSAR widgets
									• Fail: Open server logon retry dialog.
									Notes: n/a

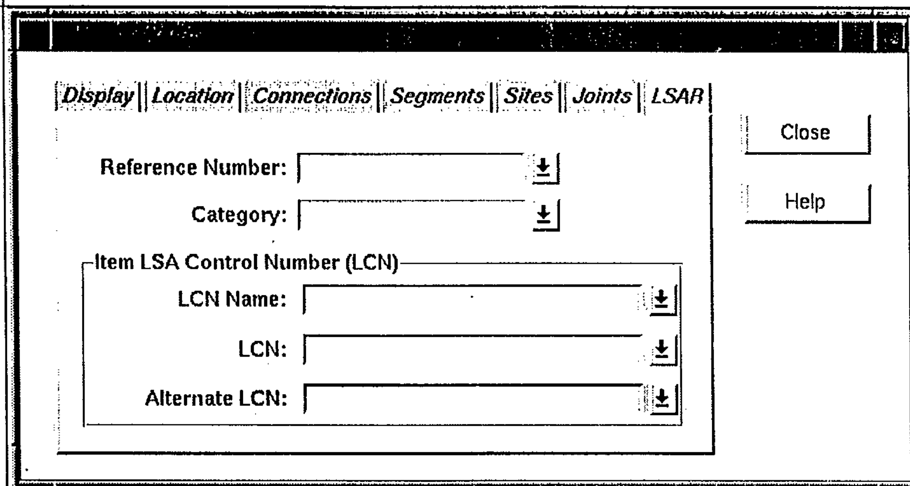
			Logon	Button					**Title: Cancel**

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Item: Cancel button to close the dialog without logging on to the server **Default:** n/a **Dependency:** n/a **Action:** Cancel Logon dialog without sending the logon command. **Notes:** n/a
			Logon Continue	Window					 **Figure 7: Logon Continue**
			Logon Continue	Entry			X	X	**Title: User Id** **Item:** User id for the account on the LSAR database server **Default:** LSAR object value **Dependency:** n/a **Action:** n/a **Notes:** Used in logon command when logon button is pressed.
			Logon Continue	Entry			X		**Title: Password** **Item:** Server account password **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Data field will not show actual password entered. Plan is to show an '*' for each character entered. The password will be kept in memory but it will be broken into pieces so it can not be detected by reading memory directly. This is used in the logon command when logon button is pressed.
			Logon Continue	Button					**Title: OK** **Item:** OK Button to send logon command **Default:** n/a **Dependency:** Userid and password fields must have values entered.

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS

									Action: Send logon command to the server.
									• Pass:
									  • If LSAR information in database does not match what is in the LSAR object, post error and open workspace setup dialog
									  • Set flag to enable LSAR widgets
									• Fail: Open server logon retry dialog.
									Notes: n/a

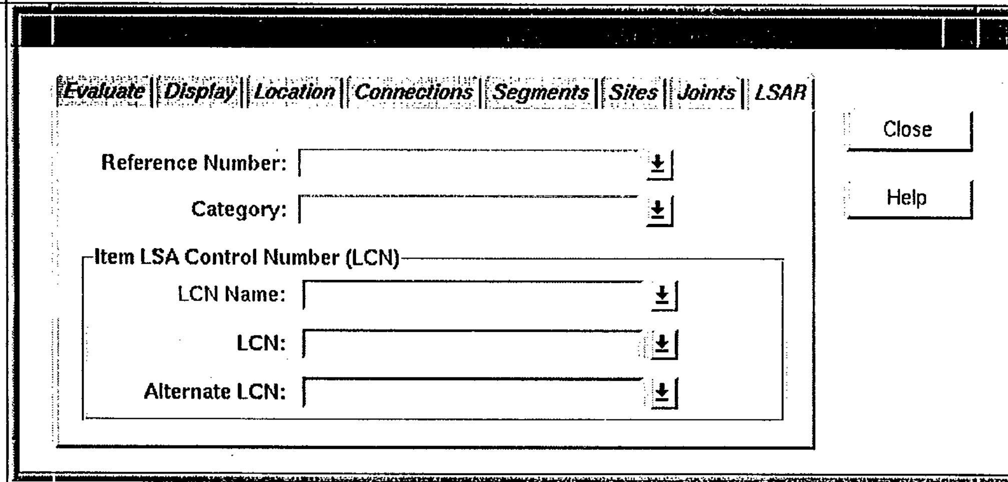
			Logon Continue	Button					**Title: Disable LSAR** **Item:** Disable LSAR button to close the dialog without logging on to the server **Default:** n/a **Dependency:** n/a **Action:** Cancel dialog without sending the logon command. Reset flag to gray LSAR widgets. **Notes:** n/a
			Logon Retry	Window					 **Figure 8: Logon Retry**
			Logon Retry	Button					**Title: Retry** **Item:** Retry button to request retry of logon **Default:** n/a **Dependency:** n/a **Action:** Return to calling dialog, Server Logon or Server Logon Continue dialog. **Notes:** n/a
			Logon Retry	Button					**Title: Cancel** **Item:** Cancel button to cancel logon retry attempt **Default:** n/a **Dependency:** n/a **Action:** Close without logging on to the server **Notes:** n/a

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
			Modify Fastener	Window					 **Figure 9: Modify Fastener**

			Modify Fastener	Button					**Title: OK**
									Item: OK button
									Default: n/a
									Dependency: n/a
									Action:
									If HA337 has changed from value in DEPTH
									• Update HA337
									• If HA183 is not 77777
									    • Update HA182
									• Else
									    • Update HA182 only if this is the first time the HA337 reference has been used.
									• Update XB203 with "P"
									• If end item part is yes
									    • Decrement HG316 and HG317 if HA337 had prior value in DEPTH
									    • Query HG316 and HG317 from LSAR
									    • Count those end item parts in fastener ring having the same HA337 value
									    • Update HG317 = (HG317 query) - (HG316 query) + count
									    • Update HG316 with this count
									If end item part changed to yes
									• Query HG316 and HG317 from LSAR

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									• Count those end item parts in fastener ring having the same HA337 value • Update HG317 = (HG317 query) - (HG316 query) + count • Update HG316 with this count If end item part changed to no • Decrement HG316 and HG317 if HA337 had prior value in DEPTH If LCN information changed and end item part is yes • Decrement HG316 and HG317 for prior LCN if HA337 had prior value in DEPTH • Query HG316 and HG317 from LSAR for new LCN • Count those end item parts in fastener ring having the same HA337 value • Update HG317 = (HG317 query) - (HG316 query) + count • Update HG316 with this count **Notes:** n/a
019	xb	2	Modify Fastener	tixComboBox	X		X	X	**Title: Alternate LCN** **Item:** Alternate logistics control code **Default:** LSAR object value else first in list from database **Dependency:** XB199 value **Action:** n/a **Notes:** Range is 00-99 but is not sequential.
199	xb	18	Modify Fastener	tixComboBox	X		X	X	**Title: LCN** **Item:** Logistics support analysis control number that represents a functional or hardware breakdow of the system. **Default:** LSAR object value else first in list from database **Dependency:** XB201 value **Action:** • Query database for XB019 list • If LSAR object value is in list, move to top of list • Set XB019 widget to value at top of list **Notes:** Generally the selection list will only be 1 item.
201	xb	19	Modify Fastener	tixComboBox					**Title: LCN Name** **Item:** Name associated with the logistics control number **Default:** Value based on key values in LSAR object else first in list from database **Dependency:** XA096 value **Action:** • Query database for XB199 list • If LSAR object value is in list, move to top of list • Set Modify Fastener XB199 widget to value at top of list **Notes:** Generally this will be 1 item.

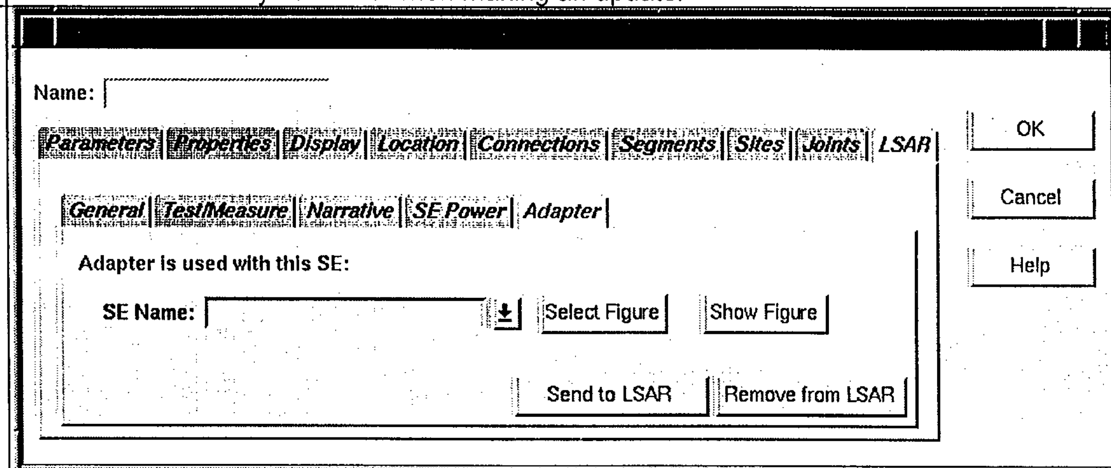
DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
203	xb	1	Modify Fastener	---	X		X	X	**Item:** Indicates if the logistic control number represents either a physical or functional breakdown. Physical means the parts exist. Functional means the parts do not exist yet. **Default:** LSAR object value else functional **Dependency:** XB019 **Action:** n/a **Notes:** Always enter "P" in database. Assumes physical part exists if HA337 exists
316	hg	4	Modify Fastener	---					**Item:** Quantity of part used per assembly. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** The user selects the assembly from the LCN name list.
317	hg	5	Modify Fastener	---					**Item:** Quantity of part used in system. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** The user selects the assembly from the LCN name list.
337	ha	32	Modify Fastener	tixComboBox	X		X	X	**Title: Reference Number** **Item:** Reference number. This is normally the manufacturer's part number. **Default:** n/a **Dependency:** n/a **Action:** When not blank, ungray end item part radio buttons, XB201, XB199, and XB019 tixComboBoxes, and HA183 tixLabelEntry **Notes:** Assume value indicates physical part which is required to make LSAR entry. Drop down shows those already in DEPTH.
177	hg	2	Modify Fastener	tixComboBox		X	X	X	**Title: Category** **Item:** Indicates how this part is used. **Default:** No **Dependency:** n/a **Action:** n/a **Notes:** Determines which table to update. End item parts are entered in HG while others are entered in CI.
			Modify Human	Button					**Title: Close** **Item:** Close button to modify a human figure in the workspace **Default:** n/a **Dependency:** n/a **Action:** If human does not have an id, query the database for the latest id. Index this by 1 for the modified human. The id ranges from A to 999.

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Notes: n/a
			Modify Tool	Window					 **Figure 10: Modify Tool**

			Modify Tool	Button					**Title: OK**
									Item: OK button
									Default: n/a
									Dependency: n/a
									Action:
									If HA337 has changed from value in DEPTH
									• Update HA337
									• If HA183 is not 77777
									    • Update HA182
									• Else
									    • Update HA182 only if this is the first time the HA337 reference has been used.
									• Update XB203 with "P"
									• If end item part is yes
									    • Decrement HG316 and HG317 if HA337 had prior value in DEPTH
									    • Query HG316 and HG317 from LSAR
									    • Count those end item parts in fastener ring having the same HA337 value
									    • Update HG317 = (HG317 query) - (HG316 query) + count
									    • Update HG316 with this count

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									If end item part changed to yes • Query HG316 and HG317 from LSAR • Count those end item parts in fastener ring having the same HA337 value • Update HG317 = (HG317 query) - (HG316 query) + count • Update HG316 with this count If end item part changed to no • Decrement HG316 and HG317 if HA337 had prior value in DEPTH If LCN information changed and end item part is yes • Decrement HG316 and HG317 for prior LCN if HA337 had prior value in DEPTH • Query HG316 and HG317 from LSAR for new LCN • Count those end item parts in fastener ring having the same HA337 value • Update HG317 = (HG317 query) - (HG316 query) + count • Update HG316 with this count **Notes:** n/a
019	xb	2	Modify Tool	tixComboBox	X		X	X	**Title: Alternate LCN** **Item:** Alternate logistics control code **Default:** LSAR object value else first in list from database **Dependency:** XB199 value **Action:** n/a **Notes:** Range is 00-99 but is not sequential.
177	hg	2	Modify Tool	tixComboBox		X	X	X	**Title: Category** **Item:** Indicates how this part is used. **Default:** No **Dependency:** n/a **Action:** n/a **Notes:** Determines which table to update. End item parts are entered in HG while others are entered in CI.
182	ha	19	Modify Tool	---		X			**Item:** Name of the part **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** This value comes from toolbox file
183	ha	5	Modify Tool	---		X			**Item:** Code for approved names **Default:** 77777 **Dependency:** n/a **Action:** n/a **Notes:** This code is used with DED 182. This value will be 77777

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
199	xb	18	Modify Tool	tixComboBox	X		X	X	**Title: LCN** **Item:** Logistics support analysis control number that represents a functional or hardware breakdow of the system. **Default:** LSAR object value else first in list from database **Dependency:** XB201 value **Action:** • Query database for XB019 list • If LSAR object value is in list, move to top of list • Set XB019 widget to value at top of list **Notes:** Generally the selection list will only be 1 item.
201	xb	19	Modify Tool	tixComboBox					**Title: LCN Name** **Item:** Name associated with the logistics control number **Default:** Value based on key values in LSAR object else first in list from database **Dependency:** XA096 value **Action:** • Query database for XB199 list • If LSAR object value is in list, move to top of list • Set Modify Tool XB199 widget to value at top of list **Notes:** Generally this will be 1 item.
203	xb	1	Modify Tool	---	X		X	X	**Item:** Indicates if the logistic control number represents either a physical or functional breakdown. Physical means the parts exist. Functional means the parts do not exist yet. **Default:** LSAR object value else functional **Dependency:** XB019 **Action:** n/a **Notes:** Always enter "P" in database. Assumes physical part exists if HA337 exists
319	cg	5	Modify Tool	---					**Item:** Number of items used to perform the task **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Use a count of the tools having the same HA337 reference for this value
319	ci	5	Modify Tool	---					**Item:** Number of parts used to perform the task **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Use a count of the tools having the same HA337 reference for this value
337	ha	32	Modify Tool	tixComboBox	X		X	X	**Title: Reference Number** **Item:** Reference number. This is normally the manufacturer's part number.

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Default: n/a **Dependency:** n/a **Action:** n/a. **Notes:** This values comes from toolbox file. Gray other widgets if this has no value in it.
491	cg	2	Modify Tool	---					**Item:** This is the unit of measure for the quantity of SE given by CG319 **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** "EA" will always be used when making an update.
491	ci	2	Modify Tool	---					**Item:** This is the unit of measure for the quantity of parts given by CI319 **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** "EA" will always be used when making an update.
			Object Adapter	Window		X			 **Figure 11: Object Adapter**
			Object Adapter	tixComboBox		X			**Title: SE Name** **Item:** Name of a support equipment figure **Default:** First support equipment figure in ring **Dependency:** n/a **Action:** n/a **Notes:** n/a

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
			Object Adapter	Button		X			**Title: Send to LSAR** **Item:** Send to LSAR button **Default:** n/a **Dependency:** n/a **Action:** Update table UB, UI, and UJ with key fields from SE and UUT selected Update EA005 to "Y" for the SE **Notes:** n/a
			Object Adapter	Button		X			**Title: Remove from LSAR** **Item:** Remove from LSAR button **Default:** n/a **Dependency:** n/a **Action:** Remove table UB, UI, and UJ entries for SE and UUT selected Update EA005 to "N" for the SE **Notes:** n/a
005	ea	1	Object Adapter	---		X			**Item:** Code indicating if adapter interconnection is required. See below for other tables that must first be completed **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Code indicates if adapter is required to connect SE to UUT.

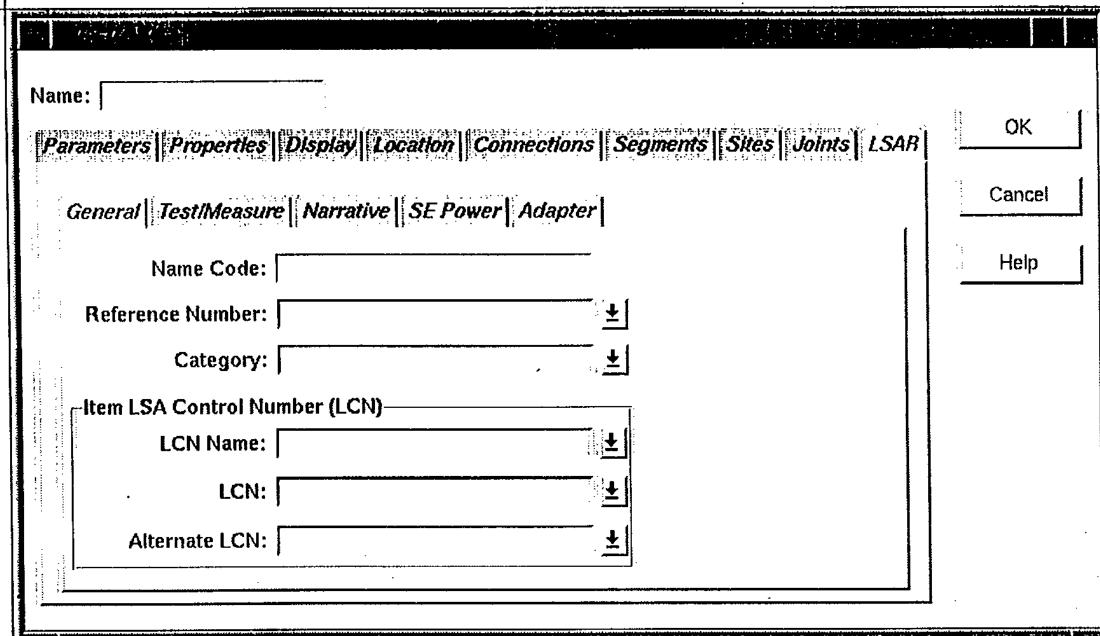
<table>
<tr><th>DED</th><th>TABLE</th><th>SIZE</th><th>DIALOG</th><th>WIDGET</th><th>K</th><th>O</th><th>M</th><th>F</th><th>COMMENTS</th></tr>
<tr><td></td><td></td><td></td><td>Object General</td><td>Window</td><td></td><td></td><td></td><td></td><td>

Figure 12: Object General</td></tr>
<tr><td></td><td></td><td></td><td>Object General</td><td>Button</td><td></td><td></td><td></td><td></td><td>Title: OK
Item: OK button
Default: n/a
Dependency: n/a
Action:
If HA337 has a value
• Update HA337
• If HA183 is not 77777
    • Update HA182
• Else
    • Update HA182 only if this is the first time the HA337 reference has been used.
• If UUT
    • Update UA with XB019, XB199, and XB203 from dialog and XA096
• Update XB203 with “P”
• Update EA177</td></tr>
</table>

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS

									• If SE
									    • Use Jack bounding box command to get dimensions
									    • Update EA268 with dimensions
									    • Update EA491 with "cm"
									    • Update EA412 with the value in HA182
									    • Update EA294 with "DEPTH"
									    • If HA182 was updated
									        • Update EA412 with HA182 name
									• If end item part is yes
									    • Query HG316 and HG317 from LSAR
									    • Count those end item parts in fastener ring having the same HA337 value
									    • Update HG317 = (HG317 query) - (HG316 query) + count
									    • Update HG316 with this count
									Notes: n/a

019	xb	2	Object General	tixComboBox	X		X	X	**Title: Alternate LCN** **Item:** Alternate logistics control code **Default:** LSAR object value else first in list from database **Dependency:** XB199 value **Action:** n/a **Notes:** Range is 00-99 but is not sequential.
177	ea	2	Object General	tixComboBox			X	X	**Title: Category** **Item:** Identifies the type and category of an item using a code. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Use to determine if object is SE, UUT, or Other for graying logic. Same widget as HG177.
177	hg	2	Object General	tixComboBox			X	X	**Title: Category** **Item:** Identifies the type and category of an item using a code. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Use to determine if object is SE, UUT, or Other for graying logic. Same widget as EA177.
182	ha	19	Object General	---		X			**Item:** Name of the part **Default:** n/a **Dependency:** HA337 **Action:** n/a

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Notes: Use the DEPTH name of the object for the first entry. If more of the same HA337 reference are made, do not change the name.
183	ha	5	Object General	tixLabelEntry		X			**Title: Name Code** **Item:** Code for approved names **Default:** 77777 **Dependency:** HA337 **Action:** n/a **Notes:** This code is used with DED 182. The default is the code for a name that is not standard. This is grayed out on the modify dialog.
199	xb	18	Object General	tixComboBox	X		X	X	**Title: LCN** **Item:** Logistics support analysis control number that represents a functional or hardware breakdow of the system. **Default:** LSAR object value else first in list from database **Dependency:** XB201 value **Action:** • Query database for XB019 list • If LSAR object value is in list, move to top of list • Set XB019 widget to value at top of list **Notes:** Generally the selection list will only be 1 item.
201	xb	19	Object General	tixComboBox					**Title: LCN Name** **Item:** Name associated with the logistics control number **Default:** Value based on key values in LSAR object else first in list from database **Dependency:** XA096 value **Action:** • Query database for XB199 list • If LSAR object value is in list, move to top of list • Set Insert Fastener XB199 widget to value at top of list **Notes:** Generally this will be 1 item.
203	xb	1	Object General	---	X		X	X	**Item:** Indicates if the logistic control number represents either a physical or functional breakdown. Physical means the parts exist. Functional means the parts do not exist yet. **Default:** LSAR object value else functional **Dependency:** XB019 **Action:** n/a **Notes:** Always enter "P" in database. Assumes physical part exists if HA337 exists
268	ea	14	Object General	---		X			**Item:** Dimensions of the SE **Default:** n/a **Dependency:** SE

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Action: n/a **Notes:** When SE is selected get the dimensions of the object. Use the JACKfigure_bbox CAPI command and calculate bounding box from results. The figure-bbox lisp command could also be used if the CAPI command does not work. DO NOT use the figure-localbbox lisp command since it may not give the information desired.
294	ea	25	Object General	---		X			**Item:** Name of activity preparing SE data **Default:** off **Dependency:** SE **Action:** n/a **Notes:** This is the name of the organization that is populating the information in table ea. In our case this is DEPTH. Always enter DEPTH in this field.

337	ha	32	Object General	tixComboBox	X		X	X	**Title: Reference Number**
									Item: Reference number. This is normally the manufacturer's part number.
									Default: n/a
									Dependency: n/a
									Action:
									• When not blank, ungray the rest of the page
									• If SE
									    • ungray Adapter, Test/Measure, Narrative, and SE Power tabs
									• If UUT
									    • ungray Test/Measure tab
									Notes: Assume value indicates physical part which is required to make LSAR entry. Drop down shows those already in DEPTH.

412	ea	42	Object General	---					**Item:** Name of the SE **Default:** n/a **Dependency:** HA182 Change **Action:** n/a **Notes:** Use the DEPTH name of the support equipment.
491	ea		Object General	---		X			**Item:** This is the unit of measure for the size of the SE given by EA268 **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** With EA268 we have a given unit of centimeter. So "CM" will always be used when making an update.

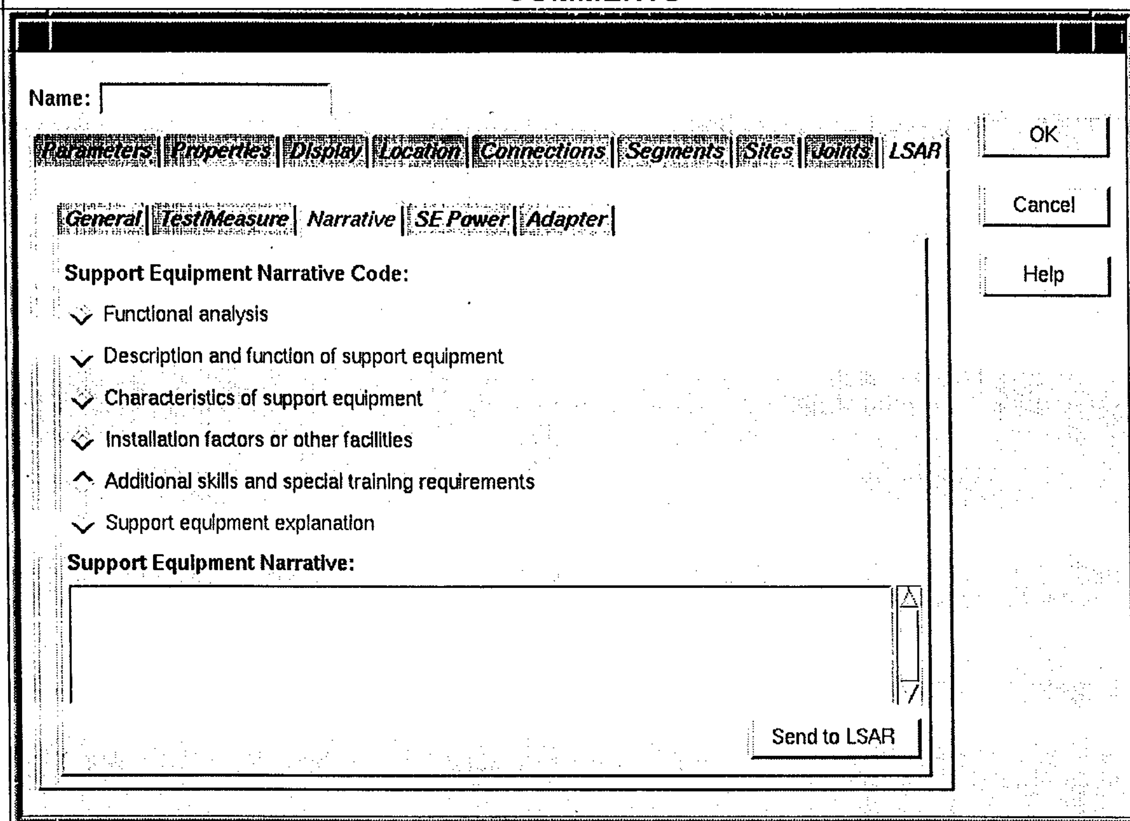
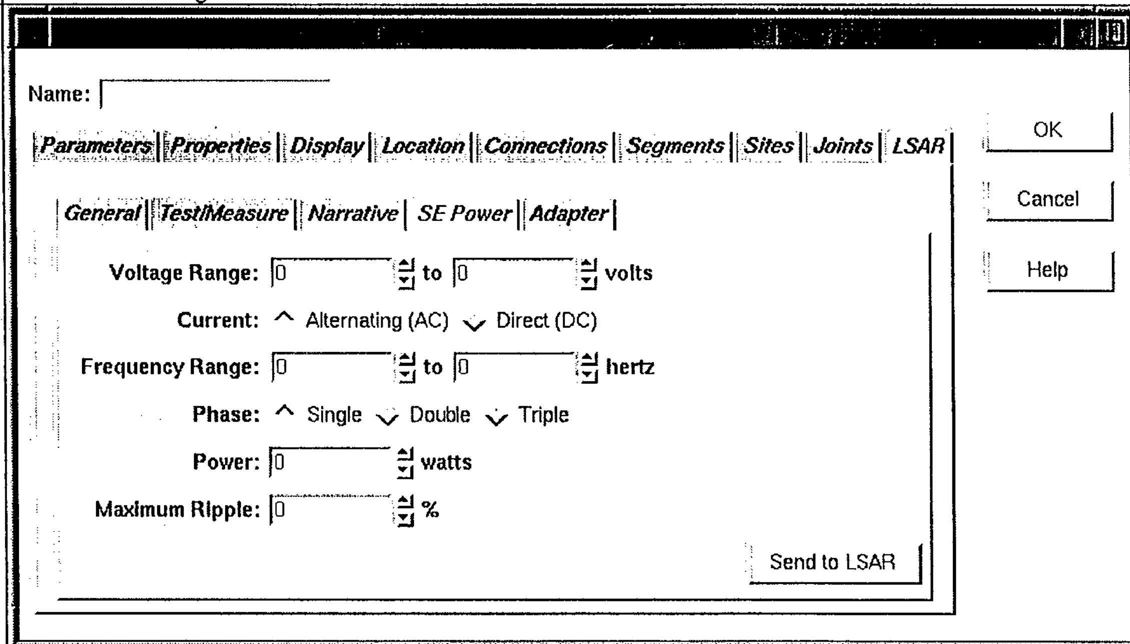
<table>
<tr><th>DED</th><th>TABLE</th><th>SIZE</th><th>DIALOG</th><th>WIDGET</th><th>K</th><th>O</th><th>M</th><th>F</th><th>COMMENTS</th></tr>
<tr><td></td><td></td><td></td><td>Object Narrative</td><td>Window</td><td></td><td>X</td><td></td><td></td><td>


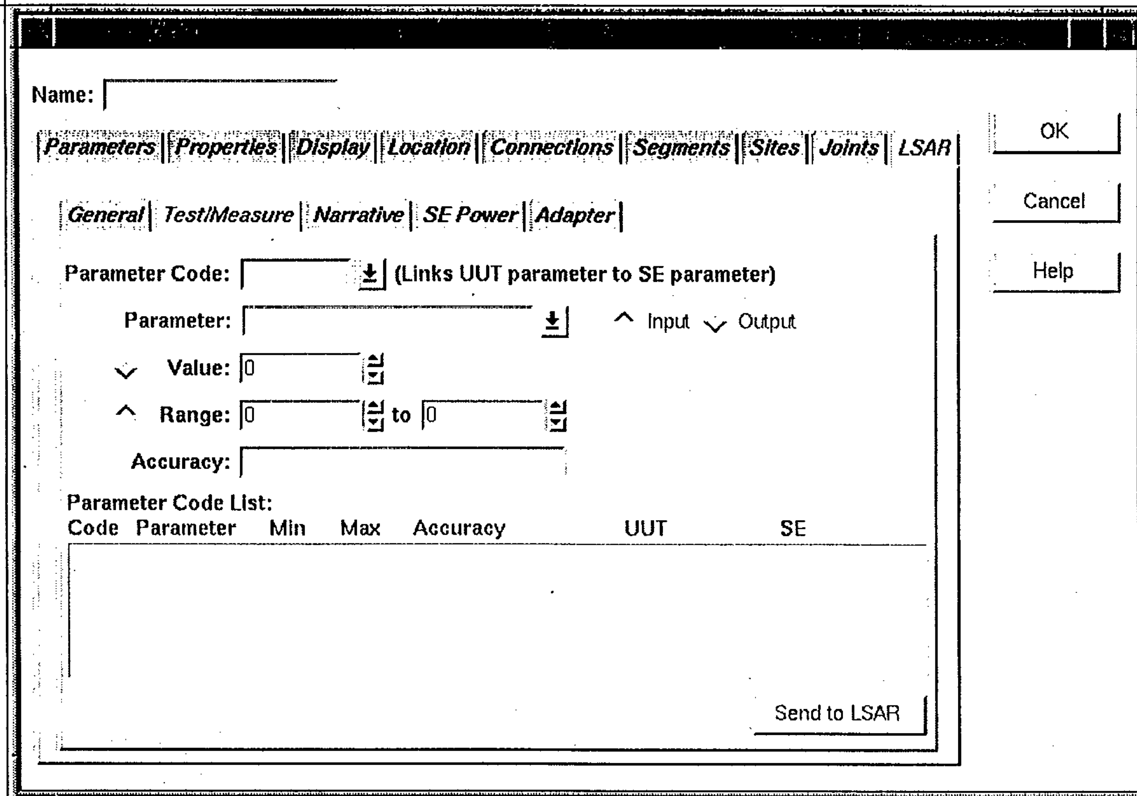
Figure 13: Object Narrative</td></tr>
<tr><td></td><td></td><td></td><td>Object Narrative</td><td>Button</td><td></td><td>X</td><td></td><td></td><td>Title: Send to LSAR
Item: Send to LSAR button
Default: n/a
Dependency: n/a
Action:
Update EE414 based on radio button selected
Update EE450 with code
Update DED selected by radio button:
EE008, EE044, EE078, EE147, EE169, EE411</td></tr>
</table>

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Notes: The range of values is from 1 to 99999. Text is appended to existing text. Must use EAGLE to remove it. When click on radio button, text field is emptied.
008	ee	65	Object Narrative	Text / Scroll		X			**Title: Support Equipment Narrative** **Item:** Describe new skills and training required to operate equipment **Default:** n/a **Dependency:** EE414 **Action:** n/a **Notes:** n/a
044	ee	65	Object Narrative	Text / Scroll		X			**Title: Support Equipment Narrative** **Item:** Describe the operational characteristics of the SE. **Default:** n/a **Dependency:** EE414 **Action:** n/a **Notes:** n/a
078	ee	65	Object Narrative	Text / Scroll		X			**Title: Support Equipment Narrative** **Item:** Describe the SE required to satisfy the functional requirements of the end article. **Default:** n/a **Dependency:** EE414 **Action:** n/a **Notes:** n/a
147	ee	65	Object Narrative	Text / Scroll		X			**Title: Support Equipment Narrative** **Item:** Describe, in technical terms, the function requiring support. **Default:** n/a **Dependency:** EE414 **Action:** n/a **Notes:** n/a
169	ee	65	Object Narrative	Text / Scroll		X			**Title: Support Equipment Narrative** **Item:** Describe requirements for installation of the SE. **Default:** n/a **Dependency:** EE414 **Action:** n/a **Notes:** n/a
411	ee	65	Object Narrative	Text / Scroll		X			**Title: Support Equipment Narrative** **Item:** Describe conditions not cover in another data element **Default:** n/a **Dependency:** EE414 **Action:** n/a

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Notes: n/a
414	ee	1	Object Narrative	Radio Button	X	X			**Title: Support Equipment Narrative Code** **Item:** Code indicating which DED is associated with the narrative. **Default:** Additional Skills and Special Training Requirements **Dependency:** n/a **Action:** n/a **Notes:** n/a
450	ee	5	Object Narrative	---	X	X			**Item:** Text sequence code. This shows the order the narrative was enter if it extends beyond one line. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** The range of values is from 1 to 99999.
			Object Power	Window		X			 **Figure 14: Object Power**
			Object	Button		X			**Title: Send to LSAR**

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
			Power						**Item:** Send to LSAR button **Default:** n/a **Dependency:** n/a **Action:** Query EI168 for identified number Increment number Update EI168 **Notes:** n/a
168	ei	3	Object Power	tixControl		X			**Title: Voltage Range** **Item:** A combination of subfields to describe the operating power requirements for the SE. This second subfield contains the min. operating voltage. **Default:** n/a **Dependency:** SE **Action:** n/a **Notes:** n/a
168	ei	3	Object Power	tixControl		X			**Title: Voltage Range** **Item:** A combination of subfields to describe the operating power requirements for the SE. This third subfield contains the max. operating voltage. **Default:** n/a **Dependency:** SE **Action:** n/a **Notes:** n/a
168	ei	3	Object Power	tixControl		X			**Title: Frequency Range** **Item:** A combination of subfields to describe the operating power requirements for the SE. This fifth subfield contains the min. frequency for a given operating voltage. **Default:** n/a **Dependency:** SE **Action:** n/a **Notes:** n/a
168	ei	3	Object Power	tixControl		X			**Title: Frequency Range** **Item:** A combination of subfields to describe the operating power requirements for the SE. This sixth subfield contains the max. frequency for a given operating voltage. **Default:** n/a **Dependency:** SE **Action:** n/a **Notes:** n/a.
168	ei	5	Object	tixControl		X			**Title: Power**

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
			Power						**Item:** A combination of subfields to describe the operating power requirements for the SE. This eighth subfield contains the power requirements in watts. **Default:** n/a **Dependency:** SE **Action:** n/a **Notes:** n/a
168	ei	4	Object Power	tixControl		X			**Title: Maximum Ripple** **Item:** A combination of subfields to describe the operating power requirements for the SE. This ninth subfield contains the percent max. ripple allowed. **Default:** n/a **Dependency:** SE **Action:** n/a **Notes:** n/a
168	ei	1	Object Power	Radio Button		X			**Title: Current** **Item:** A combination of subfields to describe the operating power requirements for the SE. This fourth subfield contains a flag for the type of voltage **Default:** AC **Dependency:** SE **Action:** n/a **Notes:** n/a.
168	ei		Object Power	Radio Button		X			**Title: Phase** **Item:** A combination of subfields to describe the operating power requirements for the SE. This seventh subfield contains the phase for an AC operating voltage. **Default:** single phase **Dependency:** SE **Action:** n/a **Notes:** n/a
168	ei	2	Object Power	---	X	X	X	X	**Item:** A combination of subfields to describe the operating power requirements for the SE. This first subfield is a unique identifier number. **Default:** n/a **Dependency:** SE **Action:** n/a **Notes:** The database will be queried for the last number assigned. This will be indexed and used here. The range is 1 to 99.

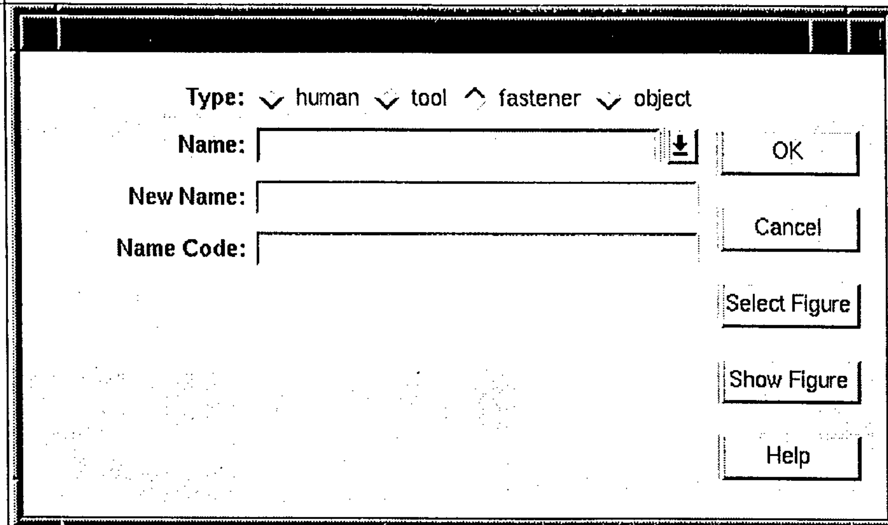
DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
			Object Test / Measure	Window		X			 **Figure 15: Object Test / Measure**
			Object Test / Measure	tixScrolledList Box		X			**Title: Parameter Code List** **Item:** List of existing codes and associated parameters from either UN or EC **Default:** n/a **Dependency:** UUT or SE **Action:** n/a **Notes:** When tab is raised If SE • Query EC284 and UN284 • Show list of values from UN284

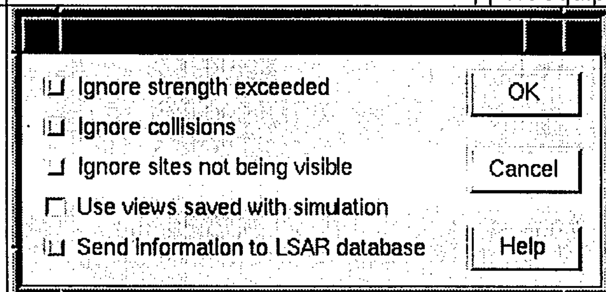
DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									• Show associated SE from EC284 If UUT • Query EC284 and UN284 • Show list of values from EC284 • Show associated UUT from UN284
			Object Test / Measure	Button		X			**Title: Send to LSAR** **Item:** Send to LSAR button **Default:** n/a **Dependency:** n/a **Action:** If SE Update EC284 If UUT Update UN284 **Notes:** n/a
284	ec	26	Object Test / Measure	tixLabelEntry		X			**Title: Accuracy** **Item:** A combination of subfields to describe the capabilities of the SE. This sixth subfield describes the tolerances or accuracy of the parameter. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** User to put in "+-", a number, and the units.
284	un	26	Object Test / Measure	tixLabelEntry		X			**Title: Accuracy** **Item:** A combination of subfields to describe the measured parameters on the UUT. This sixth subfield describes the tolerances or accuracy of the parameter. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** User to put in "+-", a number, and the units.
284	ec	10	Object Test / Measure	tixControl		X			**Title: Value and Range** **Item:** A combination of subfields to describe the capabilities of the SE. This fourth subfield has the lowest value or specific value of the parameter the SE can measure. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Add floating point box to enter a number.
284	ec	10	Object Test / Measure	tixControl		X			**Title: Range** **Item:** A combination of subfields to describe the capabilities of the SE. This fifth subfield has the highest value of the parameter the SE can measure.

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Default: n/a **Dependency:** n/a **Action:** n/a **Notes:** Add floating point box to enter a number.
284	un	10	Object Test / Measure	tixControl		X			**Title: Value and Range** **Item:** A combination of subfields to describe the measured parameters on the UUT. This fourth subfield has the lowest value or specific value of the parameter measured on the UUT. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Add floating point box to enter a number.
284	un	10	Object Test / Measure	tixControl		X			**Title: Range** **Item:** A combination of subfields to describe the measured parameters on the UUT. This fifth subfield has the highest value of the parameter measured on the UUT. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Add floating point box to enter a number.
284	ec	2	Object Test / Measure	tixComboBox	X	X			**Title: Parameter Code** **Item:** A combination of subfields to describe the capabilities of the SE. This first subfield associates an SE parameter to an UUT. **Default:** First name in list **Dependency:** n/a **Action:** Query UN284 for this code If existing code • Enter UN284 parameter in parameter widget and gray Else • Ungray parameter widget for user entry **Notes:** This is a code assigned to an SE parameter. The same code is used for the associated UUT parameter. When SE, a drop down box will show the existing UUT parameters to select the one that matches. The code from the UUT will be used here. If there is no match, select NEW. The database will be queried for the existing codes used for both SE and UUT. The highest code will be indexed to the next one and used here. Since there may be multiple parameters, this selection process can be repeated several times.
284	ec	12	Object Test / Measure	tixComboBox		X			**Title: Parameter** **Item:** A combination of subfields to describe the capabilities of the SE. This third subfield identifies

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									the parameter (e.g. volts, Hertz, etc.). **Default:** volts **Dependency:** n/a **Action:** n/a **Notes:** Add a data entry field the user can fill in with drop down to select from most often used parameters.
284	un	2	Object Test / Measure	tixComboBox	X	X			**Title: Parameter Code** **Item:** A combination of subfields to describe the measured parameters on the UUT. This first subfield associates an UUT parameter to an SE. **Default:** First name in list **Dependency:** n/a **Action:** Query EC284 for this code If existing code • Enter EC284 parameter in parameter widget and gray Else • Ungray parameter widget for user entry **Notes:** This is a code assigned to an UUT parameter. The same code is used for the associated SE parameter. When UUT, a drop down box will show the existing SE parameters to select the one that matches. The code from the SE will be used here. If there is no match, select NEW. The database will be queried for the existing codes used for both SE and UUT. The highest code will be indexed to the next one and used here. Since there may be multiple parameters, this selection process can be repeated several times.
284	un	12	Object Test / Measure	tixComboBox		X			**Title: Parameter** **Item:** A combination of subfields to describe the measured parameters on the UUT. This third subfield identifies the parameter (e.g. volts, Hertz, etc.). **Default:** volts **Dependency:** n/a **Action:** n/a **Notes:** Add a data entry field the user can fill in with drop down to select from most often used parameters.
284	ec	1	Object Test / Measure	radiobutton		X			**Title: Value and Range** **Item:** A combination of subfields to describe the capabilities of the SE. This seventh subfield specifies if the parameter is a range or specific value. **Default:** range **Dependency:** n/a **Action:** n/a

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Notes: n/a
284	un	1	Object Test / Measure	radiobutton		X			**Title: Value and Range** **Item:** A combination of subfields to describe the measured parameters on the UUT. This seventh subfield specifies if the parameter is a range or specific value. **Default:** range **Dependency:** n/a **Action:** n/a **Notes:** n/a
284	ec	1	Object Test / Measure	Radio Button		X			**Title: Input / Output** **Item:** A combination of subfields to describe the capabilities of the SE. This second subfield indicates if the parameter is an input to the SE or an output from the SE. **Default:** input **Dependency:** n/a **Action:** n/a **Notes:** n/a
284	un	1	Object Test / Measure	Radio Button		X			**Title: Input / Output** **Item:** A combination of subfields to describe the measured parameters on the UUT. This second subfield indicates if the parameter is an input to the UUT or an output from the UUT. **Default:** input **Dependency:** n/a **Action:** n/a **Notes:** n/a

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
			Rename Figure	Window					 **Figure 16: Rename Figure**
			Rename Figure	Button					**Title: OK** **Item:** OK button **Default:** n/a **Dependency:** n/a **Action:** If HA183 is not 77777 • Update HA182 If HA182 has changed and this is first figure to use HA337 reference • Update HA182 If EA412 has value and HA182 was changed • Update EA412 with the same value as HA182 **Notes:** n/a
182	ha	19	Rename Figure	tixLabelEntry		X			**Title: New Name** **Item:** Name of the part **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** If the HA183 value is not 77777, then update HA182

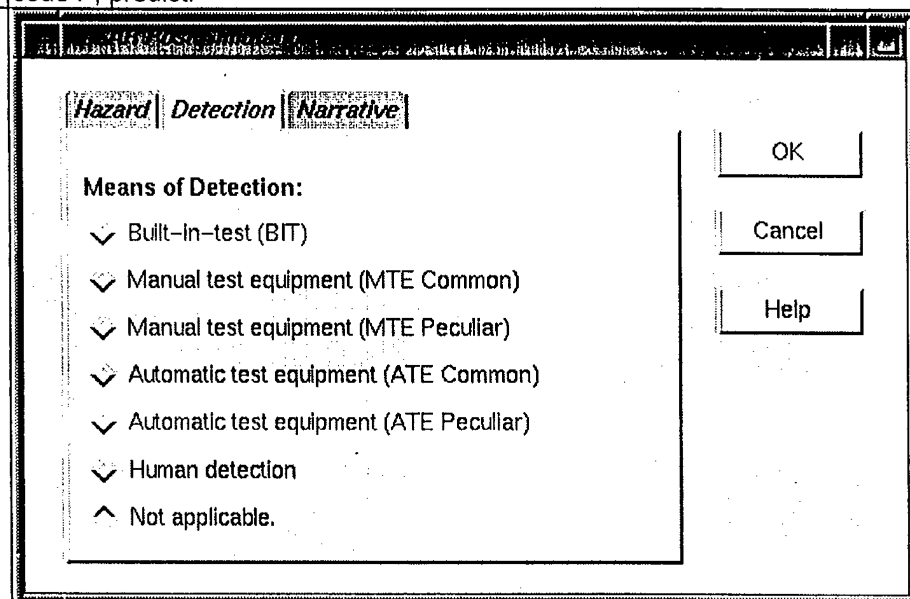
DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
183	ha	5	Rename Figure	tixLabelEntry		X			**Title: Name Code** **Item:** Code for approved names **Default:** 77777 **Dependency:** n/a **Action:** n/a **Notes:** This code is used with DED 182. The default is the code for a name that is not standard.
412	ea	42	Rename Figure	---					**Item:** Name of the SE **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Use the DEPTH name of the support equipment.
			Run Simulation	Window					 **Figure 17: Run Simulation**
			Run Simulation	checkbox					**Title: Send information to LSAR database** **Item:** Send simulation run information to LSAR database **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** n/a
			Run Simulation	button					**Title: OK** **Item:** OK button **Default:** n/a **Dependency:** n/a **Action:** If checkbox is active to send information to LSAR • After subtask completes • Update CB227 with subtask time

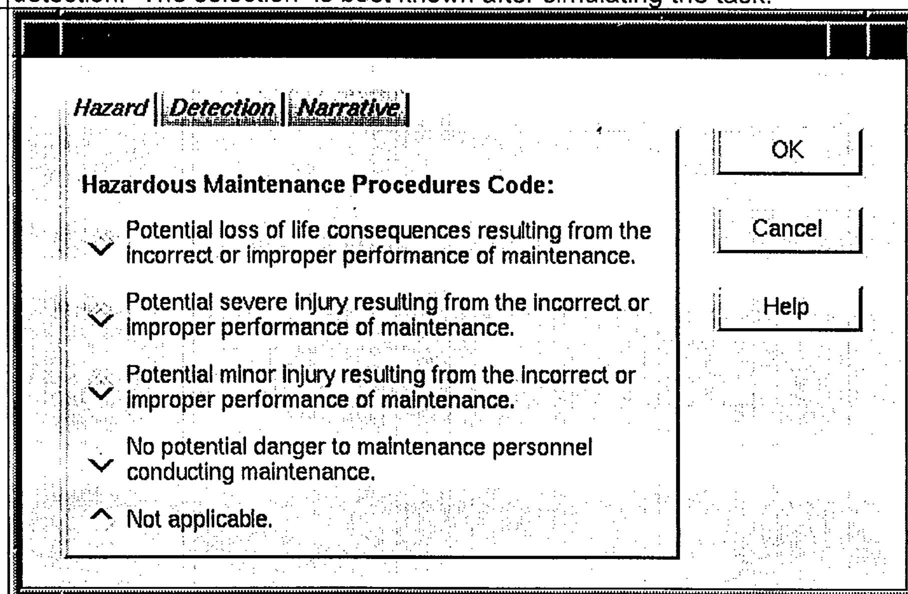
DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS

									• After task completes
									  • Open Simulation Hazards dialog for user input of tabbed dialogs
									  • Update CA224 with task time
									  • Update CA225 with (task time) X (crew size)
									  • Update XB342 with "Y" before making entries in BD
									    • Update BD222 with maximum time to repair
									    • Update BD286 with percentile
									    • Update BD347 with "P"
									• After simulation completes
									  • Update AA064 with crew size if value is larger than what is in database
									  • Update AA222 with maximum time to repair (sum of BD222 values)
									  • Update AA286 with percentile (same as BD286)
									Notes: n/a

064	aa	4	Run Simulation	---					**Item:** This is the number of personnel assigned to operate the system. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Use method Figures[type].size() to get the number of humans in the workspace. This assumes all humans in the workspace at the time of simulation make up the crew for the system. This assumes the simulation covers the whole system and people are not added and deleted for various tasks.
222	aa	5	Run Simulation	---					**Item:** The max. corrective maintenance downtime within which a specified percent of all corrective maintenance actions can be accomplished. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** This needs to be calculated from the times returned by the motion models. Collect the times returned for all subtasks and tasks making up a maintenance action. Add these times together to get the time to enter into the database.
222	bd	5	Run Simulation	---					**Item:** The required/specified max. corrective maintenance downtime within which a specified percent of all corrective maintenance actions can be accomplished. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** This needs to be calculated from the times returned by the motion models. Collect the times returned for all subtasks and tasks making up a maintenance action. Add these times together to get the time to enter into the database.

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
224	ca	5	Run Simulation	---					**Item:** This is the estimated time required to perform a task in hours. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** At an end of task the times returned by the motion models can be added together to get this number.
225	ca	5	Run Simulation	---					**Item:** This is the man hours estimated to do a task. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Track the number of humans required to perform a task during the simulation run. Multiply this number times the value from CA224.
226	cd	4	Run Simulation	---					**Item:** The time in minutes to complete a subtask by person. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** For each motion model we need to track which humans do which task elements. Keep a running total for each one for the minutes to do the whole subtask. Update the table with the values for each person by id.
227	cb	5	Run Simulation	---					**Item:** This is the estimated time required to perform a subtask in minutes. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** At an end of subtask the times returned by the motion models can be added together to get this number.
286	aa	2	Run Simulation	---					**Item:** The percentage of all corrective maintenance actions that can be accomplished within a specified maximum time to repair.. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** All times returned to us from Ranko will have a tolerance. This tolerance should be the same for all the times. Using this tolerance we can make a fixed entry in this field. For example, if the tolerance on Ranko's times is +- 5% we may want to always enter 95% in this field.
286	bd	2	Run Simulation	---					**Item:** The required/specified percentage of all corrective maintenance actions that can be accomplished within a specified maximum time to repair. **Default:** n/a **Dependency:** n/a

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Action: n/a **Notes:** All times returned to us from Ranko will have a tolerance. This tolerance should be the same for all the times. Using this tolerance we can make a fixed entry in this field. For example, if the tolerance on Ranko's times is +- 5% we may want to always enter 95% in this field.
342	xb	1	Run Simulation	---					**Item:** Code indicates if RAM information is included for the LCN. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** This must be "Y" if table bd is to be completed.
347	bd	1	Run Simulation	---	X				**Item:** Code to indicate if values are allocated, predicted, or measured. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Since our values will be from a table and not actual measurements we will always use the code P, predict.
			Simulation Detection	Window		X			 **Figure 18: Simulation Detection**
			Simulation	button		X			**Title: OK**

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
			Detection						**Item:** OK button **Default:** n/a **Dependency:** n/a **Action:** Update CA237 **Notes:** n/a
237	ca	1	Simulation Detection	radiobutton		X			**Title: Means of Detection** **Item:** Code to indicate how anything from subassembly to a system is tested to verify its operation during the task. **Default:** Not applicable **Dependency:** n/a **Action:** n/a **Notes:** The type of test run on an UUT may be dependent on the task so this code can not be entered when the UUT CAD is imported. Test selections range from built-in-test to human detection. The selection is best known after simulating the task.
			Simulation Hazards	Window		X			 **Figure 19: Simulation Hazards**
			Simulation Hazards	button		X			**Title: OK** **Item:** OK button

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Default: n/a **Dependency:** n/a **Action:** Update CA155 **Notes:** n/a
155	ca	1	Simulation Hazards	radiobutton		X			**Title: Hazardous Maintenance Procedures Code** **Item:** Code to indicate if personnel will be exposed to hazardous conditions. The choice is loss of life, severe injury, minor injury, or no danger. **Default:** Value in LSAR database first then Not applicable **Dependency:** n/a **Action:** n/a **Notes:** n/a

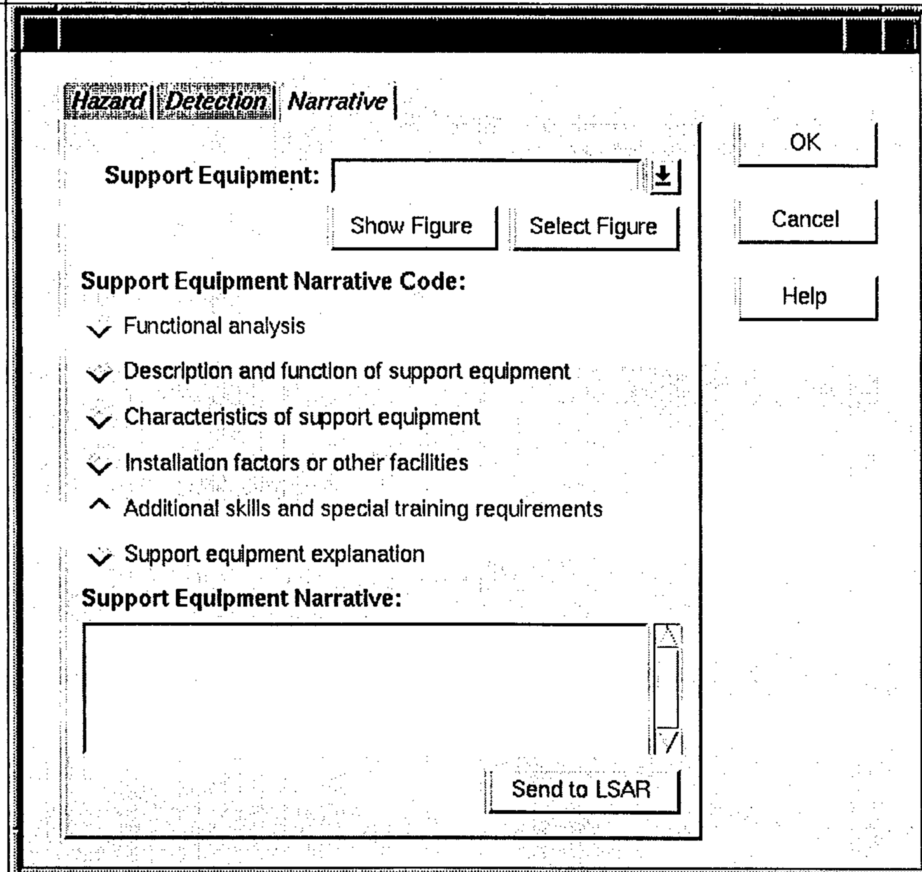
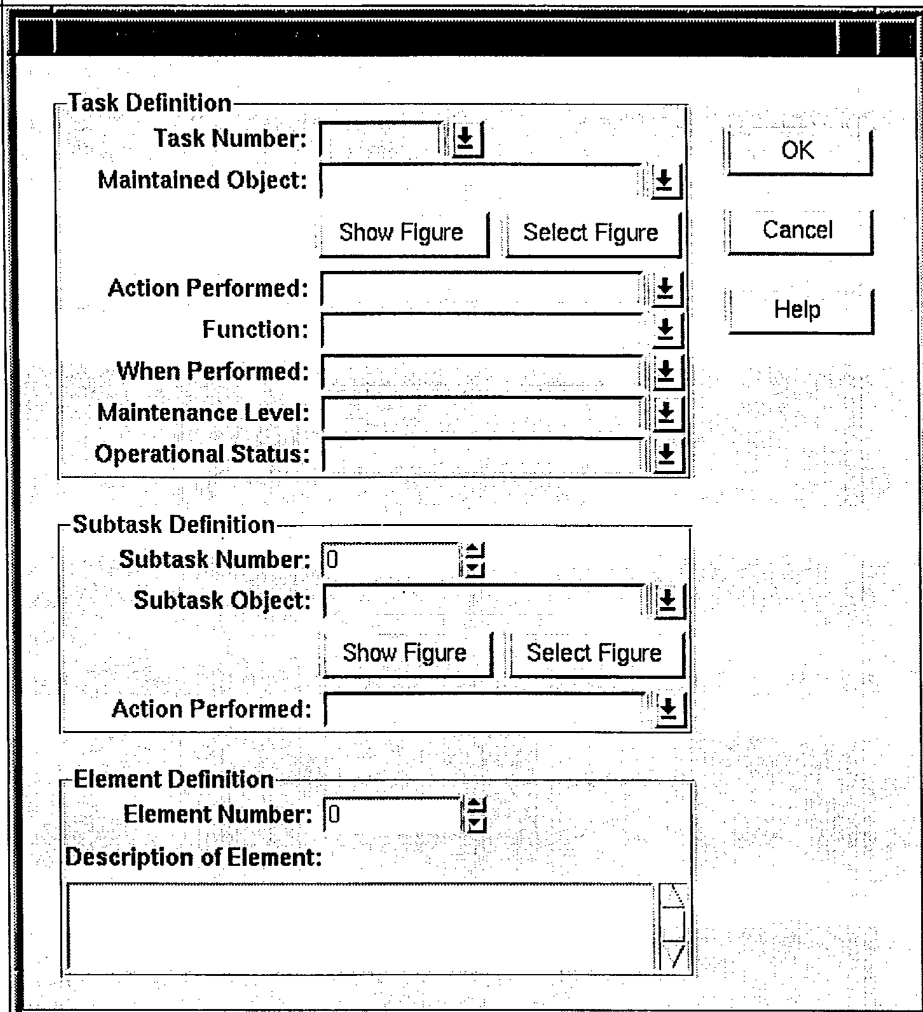
<table>
<tr><th>DED</th><th>TABLE</th><th>SIZE</th><th>DIALOG</th><th>WIDGET</th><th>K</th><th>O</th><th>M</th><th>F</th><th>COMMENTS</th></tr>
<tr><td></td><td></td><td></td><td>Simulation Narrative</td><td>Window</td><td></td><td>X</td><td></td><td></td><td>

Figure 20: Simulation Narrative</td></tr>
<tr><td></td><td></td><td></td><td>Simulation Narrative</td><td>tixComboBox</td><td></td><td>X</td><td></td><td></td><td>Title: Support Equipment
Item: Support Equipment the narrative is about
Default: First SE in list of objects
Dependency: n/a
Action: n/a
Notes: n/a</td></tr>
<tr><td></td><td></td><td></td><td>Simulation</td><td>Button</td><td></td><td>X</td><td></td><td></td><td>Title: Send to LSAR</td></tr>
</table>

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
			Narrative						**Item:** Send to LSAR button **Default:** n/a **Dependency:** n/a **Action:** Update EE414 based on radio button selected Update EE450 with code Update DED selected by radio button: EE008, EE044, EE078, EE147, EE169, EE411 **Notes:** The range of values is from 1 to 99999. Text is appended to existing text. Must use EAGLE to remove it. When click on radio button, text field is emptied.
008	ee	65	Simulation Narrative	Text / Scroll		X			**Title: Support Equipment Narrative** **Item:** Describe new skills and training required to operate equipment **Default:** n/a **Dependency:** EE414 **Action:** n/a **Notes:** n/a
044	ee	65	Simulation Narrative	Text / Scroll		X			**Title: Support Equipment Narrative** **Item:** Describe the operational characteristics of the SE. **Default:** n/a **Dependency:** EE414 **Action:** n/a **Notes:** n/a
078	ee	65	Simulation Narrative	Text / Scroll		X			**Title: Support Equipment Narrative** **Item:** Describe the SE required to satisfy the functional requirements of the end article. **Default:** n/a **Dependency:** EE414 **Action:** n/a **Notes:** n/a
147	ee	65	Simulation Narrative	Text / Scroll		X			**Title: Support Equipment Narrative** **Item:** Describe, in technical terms, the function requiring support. **Default:** n/a **Dependency:** EE414 **Action:** n/a **Notes:** n/a
169	ee	65	Simulation Narrative	Text / Scroll		X			**Title: Support Equipment Narrative** **Item:** Describe requirements for installation of the SE. **Default:** n/a

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Dependency: EE414 **Action:** n/a **Notes:** n/a
411	ee	65	Simulation Narrative	Text / Scroll		X			**Title: Support Equipment Narrative** **Item:** Describe conditions not cover in another data element **Default:** n/a **Dependency:** EE414 **Action:** n/a **Notes:** n/a
414	ee	1	Simulation Narrative	Radio Button	X	X			**Title: Support Equipment Narrative Code** **Item:** Code indicating which DED is associated with the narrative. **Default:** Additional Skills and Special Training Requirements **Dependency:** n/a **Action:** n/a **Notes:** n/a
450	ee	5	Simulation Narrative	---	X	X			**Item:** Text sequence code. This shows the order the narrative was enter if it extends beyond one line. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** The range of values is from 1 to 99999.

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
			Simulation Setup	Window					 Figure 21: Simulation Setup
			Simulation Setup	button					**Title:** OK **Item:** OK button **Default:** n/a

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Dependency: n/a **Action:** Update CA427, CA431, CB407, CB431, CC450, CC372, CC095 Check DEPTH for EA177 value Update CA358 and CA428 based on EA177 values **Notes:** n/a
095	cc	1	Simulation Setup	---		X			**Item:** Code to indicate if the narrative is for a task element. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** This is a flag used to id task elements vs. continued text, notes, warnings, and cautions. Use of this flag will be determined by the text entered in the CC372.
358	ca	3	Simulation Setup	---		X			**Item:** This contains requirements codes. We will only fill in the part for the tool/SE code. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** This indicates if the tools or SE used in the task is common, peculiar, or both. This value will be based on selections made for EA177.
372	cc	65	Simulation Setup	Text / Scroll		X			**Title: Description of Element** **Item:** Description of task element **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Text can include notes, cautions, and warnings. The field is 65 characters but continuation text is allowed.
407	cb	3	Simulation Setup	tixControl	X		X	X	**Title: Subtask Number** **Item:** A sequential number showing the order of the subtasks to perform a task. This will be handled by querying the database for the last code used. The range is 001 to 999. **Default:** Next number available **Dependency:** n/a **Action:** If existing number selected • Query database and use values to complete other fields in subtask definition **Notes:** n/a
427	ca	2	Simulation Setup	tixComboBox	X		X	X	**Title: Task Number** **Item:** A combination of 6 subfields that form the task code. This sixth subfield is a sequence code to make the task code unique.

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Default: Next number available **Dependency:** n/a **Action:** If existing number selected • Query database and use values to complete other fields in task definition **Notes:** The range is aa to zz followed by 00 to 99.
427	ca	1	Simulation Setup	tixComboBox			X	X	**Title: Function** **Item:** A combination of 6 subfields that form the task code. This first subfield is the function necessary to the operation or maintenance of an item. **Default:** First word in list **Dependency:** n/a **Action:** n/a **Notes:** When setting up the motion models the user needs to identify the action to be performed by the task. The DED offers an action word selection list to choose from such as adjust, clean, or remove. When selected, the word is changed to the corresponding code used in the database.
427	ca	1	Simulation Setup	tixComboBox			X	X	**Title: When Performed** **Item:** A combination of 6 subfields that form the task code. This second subfield identifies the timing for when the task needs to be performed. **Default:** scheduled **Dependency:** n/a **Action:** n/a **Notes:** When setting up the motion models the user needs to identify when the task needs to be performed. The DED offers a word selection list to choose from such as daily, normal, emergency. When selected, the word is changed to the corresponding code used in the database.
427	ca	1	Simulation Setup	tixComboBox			X	X	**Title: Maintenance Level** **Item:** A combination of 6 subfields that form the task code. This third subfield identifies the level of the crew authorized to perform the task such as operator, depot, or afloat. **Default:** On Equipment **Dependency:** n/a **Action:** n/a **Notes:** When setting up the motion models the user needs to identify the level of the crew authorized to perform the task. Much of this depends on where the task will be performed which should be known by the user at this time. When selected, the word is changed to the corresponding code used in the database.
427	ca	1	Simulation Setup	tixComboBox			X	X	**Title: Operational Status** **Item:** A combination of 6 subfields that form the task code. This fifth subfield identifies the operational status and mission readiness of the item during the task.

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									Default: System Inoperable during Equipment Maintenance **Dependency:** n/a **Action:** n/a **Notes:** When setting up the motion models the user needs to identify degree of incapacity of the UUT. This should be known by the user when setting up the motion models.
427	ca	1	Simulation Setup	---			X	X	**Item:** A combination of 6 subfields that form the task code. This fourth subfield identifies the service that has control over the task such as Army or Air Force. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Use value from AA376
428	ca	3	Simulation Setup	---		X			**Item:** This indicates any special considerations that must be considered during analysis of the task. We will only indicate if a special tool is required. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Value comes from selection made in EA177.
431	ca	36	Simulation Setup	tixComboBox		X			**Title: Maintained Object and Action Performed** **Item:** This is the task identifier or title. It is a 36 character max. field consisting of noun, verb, and modifiers. **Default:** Active object **Dependency:** n/a **Action:** n/a **Notes:** Selects the UUT name. Use the verb from the task code, CA427. Put the verb and name together to form this entry.
431	cb	36	Simulation Setup	tixComboBox		X			**Title: Subtask Object and (Subtask Definition) Action Performed** **Item:** This is the subtask identifier or title. It is a 36 character max. field consisting of noun, verb, and modifiers. **Default:** n/a **Dependency:** n/a **Action:** n/a **Notes:** Select object and action in 2 widgets. These are put together to form this entry.
450	cc	5	Simulation Setup	tixControl	X		X	X	**Title: Element Number** **Item:** This number shows the sequence of the task elements within the task. **Default:** Next value available **Dependency:** n/a **Action:**

DED	TABLE	SIZE	DIALOG	WIDGET	K	O	M	F	COMMENTS
									If existing number selected • Query database and use values to complete other fields in subtask definition **Notes:** The range of values is from 1 to 99999. Updates need to be made each time since task elements may change in sequence.

Table 2: Details for New and Modified Dialog Boxes

4. Data Members

DED	TABLE	CLASS	TYPE	O	M	F	COMMENTS
019	xb	Fastener	char altlcn[3]		X	X	Alternate logistics control code
177	hg	Fastener	char category[3]	X	X	X	Identifies the type and category of an item using a code.
203	xb	Fastener	char lcntype		X	X	Indicates if the logistic control number represents either a physical or functional breakdown. Physical means the parts exist. Functional means the parts do not exist yet.
199	xb	Fastener	String lcn		X	X	Logistics support analysis control number that represents a functional or hardware breakdown of the system.
337	ha	Fastener	String refNum		X	X	Reference number. This is normally the manufacturer's part number.
288	cd	Human	char lsarID[4]		X	X	This is the id for a human.
019	xb	LSAR	char altlcn[3]		X	X	Alternate logistics control code
096	xa	LSAR	char endItem[11		X	X	End item acronym code that uniquely identifies the system (e.g. TOW, Sparrow)
203	xb	LSAR	char lcntype		X	X	Indicates if the logistic control number represents either a physical or functional breakdown. Physical means the parts exist. Functional means the parts do not exist yet.
376	aa	LSAR	char service		X	X	Service designator code identifying the military or agency in charge.
		LSAR	int state		X	X	State of LSAR function, enabled or disabled
		LSAR	String database		X	X	Name of the LSAR database to access.
199	xb	LSAR	String lcn		X	X	Logistics support analysis control number that represents a functional or hardware breakdown of the system.
		LSAR	String password		X		Server account password
		LSAR	String server		X	X	Name of the LSAR database server.
		LSAR	String uid		X	X	Userid for the account on the LSAR database server
019	xb	Object	char altlcn[3]		X	X	Alternate logistics control code
177	ea	Object	char category[3]		X	X	Identifies the type and category of an item using a code.
203	xb	Object	char lcntype		X	X	Indicates if the logistic control number represents either a physical or functional breakdown. Physical means the parts exist. Functional means the parts do not exist yet.
168	ei	Object	char pwrID[3]	X	X	X	Number to make the combination of subfields describing the operating power requirements for the SE unique.
199	xb	Object	String lcn		X	X	Logistics support analysis control number that represents a functional or hardware breakdown of the system.
337	ha	Object	String refNum		X	X	Reference number. This is normally the manufacturer's part number.
407	cb	Simulation File	char subtaskNum[4]		X	X	A sequential number showing the order of the subtasks to perform a task. This will be handled by querying the database for the last code used. The range is 001 to 999.
427	ca	Simulation File	char taskcode[8]		X	X	Collection related codes used to describe the task.

DED	TABLE	CLASS	TYPE	O	M	F	COMMENTS
450	cc	Simulation File	char textSeqCode[6]		X	X	This number shows the sequence of the text used to define elements within the task.
019	xb	Tool	char altlcn[3]		X	X	Alternate logistics control code
177	hg	Tool	char category[3]	X	X	X	Identifies the type and category of an item using a code.
203	xb	Tool	char lcntype		X	X	Indicates if the logistic control number represents either a physical or functional breakdown. Physical means the parts exist. Functional means the parts do not exist yet.
199	xb	Tool	String lcn		X	X	Logistics support analysis control number that represents a functional or hardware breakdown of the system.
337	ha	Tool	String refNum		X	X	Reference number. This is normally the manufacturer's part number.

Table 3: Data Members

5. Definitions

maintenance action:	One or more tasks required to complete the action (e.g. Service vehicle)
SE:	Support equipment
step:	Smallest logically and reasonably definable unit of behavior required in completing a task or subtask (e.g. Apply torque to lug nuts with wrench)
subtask:	Activities which fulfill a portion of the immediate purpose within a task (e.g. Remove lug nuts)
task:	Composite of related activities performed for an immediate purpose (e.g. Change tire)
UUT	Unit under test

6. Change History

05-12-97	Added Modify Tool window and related DED items
05-12-97	Added Insert Tool window and revised DED items to match
05-12-97	Revised Insert Fastener window
05-12-97	Replace End Item with Category on Object General window
05-12-97	Replaced End Item with Category on Modify Fastener window
05-12-97	Remove UUT selction on Object Adapter window
05-13-97	Added data member table